UNIVERSAL LIBRARY

FUNCTION REFERENCE

ComputerBoards, Inc.
Revision 5.1
February, 2000

Notice

ComputerBoards, Inc. does not authorize any ComputerBoards, Inc. product for use in life support systems and/or devices without the written approval of the President of ComputerBoards, Inc. Life support devices/systems are devices or systems which, a) are intended for surgical implantation into the body, or b) support or sustain life and whose failure to perform can be reasonably expected to result in injury. ComputerBoards, Inc. products are not designed with the components required, and are not subject to the testing required to ensure a level of reliability suitable for the treatment and diagnosis of people.

Table of Contents

1 FUNCTION OVERVIEW

1.1 INTRODUCTION

A complete, detailed description of all Universal Library functions is included in a seperate manual included in this package. The following section provide a brief description and will provide the user with a general idea of the capability of Universal Library, and the functions that are available. We also highly recommend that you refer to one of the many example programs provided. These programs are often provide the ideal description of the various functions as well as providing you with a starting point from which to write your own programs.

1.2 ANALOG I/O FUNCTIONS

The Analog function names all begin with cbA. These functions perform analog input and output and convert analog data.

cbAIn() - Single Analog input
Takes a single reading from an analog input channel (A/D).

cbAInScan()- Analog Input Scan
Repeatedly scans a range of analog input (A/D) channels.   The channel range, the number of iterations, the sampling rate, and the A/D range can all be specified. The data that is collected is stored in an array.

cbALoadQueue() - Load Chan/Gain queue
Loads a series of chan/gain pairs into A/D board's queue. These chan/gains will be used with all subsequent analog input functions.

cbAOut() - Single analog output
Outputs a single value to an analog output (D/A).

cbAOutScan() - Analog output scan
Repeatedly scans a range of analog output (D/A) channels.   The channel range, the number of iterations, and the rate can all be specified. The data values from consecutive elements of an array are sent to each D/A channel in the scan.

cbAPretrig() - Analog pre-triggered input
Repeatedly scans a range of analog input (A/D) channels waiting for a trigger signal. When a trigger occurs it returns the specified number of samples and points before and after the trigger occurred. The channel range, the sampling rate, and the A/D range can all be specified. All of the data that is collected is stored in an array.

cbATrig() - Analog trigger
Reads the analog input and waits until it goes above or below a specified threshold. When the trigger condition is met the current sample is returned.

cbFileAInScan() - Analog input to a file

cbFilePretrig() - Pre-triggered analog input to a file

cbAConvertData() - Converts analog data from data plus channel tags to separate data and channel tags
Each raw sample from analog input is a 16 bit value. On some 12 bit A/D boards it consists of a 12 bit A/D value along with a four bit channel number. This function is not intended to be used with 16 bit A/D boards.

This conversion is done automatically by the cbAIn() function. It can also be done automatically by the cbAInScan() function with the CONVERTDATA option. In some cases though, it may be useful or necessary to collect the data and then do the conversion sometime later. The cbAConvertData() function takes a buffer full of unconverted data and converts it.

cbACalibrateData() - Calibrates analog data
Each raw sample from a board with software calibration factors which must be applied to the sample may be acquired and calibrated then passed to an array, or acquired then passed to the array without calibration. This function applies the calibration factors to an array of data after the acquisition is complete. The only case where you would withhold calibration until after the acquisition run was complete is on slower CPUs or when processing time is at a premium. Applying calibration factors in real time on a per sample basis does eat up machine cycles. To disable the automatic calibration so that you may apply the calibration later, specify the NOCALIBRATE option when collecting data with cbAInScan().

**cbAConvertPreTrigData(**) - Convert pre-trigger data
When data is collected with the cbAPretrig() function, the same conversion needs to be done as described above for cbAConvertData. There is a further complication because cbAPretrig() collects analog data into an array. It treats the array like a circular buffer. While it is waiting for the trigger to occur it fills the array. When it gets to the end it resets to the start and begins again. When the trigger signal occurs it continues collecting data into the circular buffer until the requested number of samples have been collected.

When the data acquisition is complete all of the data is in the array but it is in the wrong order. The first element of the array does not contain the first data point. The data has to be rotated in the correct order.

This conversion can be done automatically by the cbAPretrig() function with the CONVERTDATA option. In some cases though, it may be useful or necessary to collect the data and then do the conversion sometime later. The cbAConvertPretrigData() function takes a buffer full of unconverted data and converts it.

1.3 DIGITAL I/O FUNCTIONS

The digital function names all begin with "cbD". These functions perform digital input and output. They operate on various types of digital I/O ports.

cbDBitIn() - Digital bit input
Reads a single bit from a digital input port.

cbDBitOut() - Digital bit output
Sets a single bit on a digital output port.

cbDConfigPort() - Configures digital outputs
Selects whether a digital port is an input or an output.

cbDIn() - Digital single input
Reads a specified digital input port.

cbDInScan() - Digital multiple input
Reads a specified number of bytes or words from a digital input port at a specified rate.

cbDOut() - Digital single output
Writes a byte to a digital output port.

cbDOutScan() - Digital multiple output
Writes a series of bytes or words to a digital output port at a specified rate.

1.4 TEMPERATURE INPUT FUNCTIONS

The temperature sensor function names begin with "cbT". These functions convert a raw analog input from an EXP or other temperature sensor board to temperature.

cbTIn() - Single temperature input
Reads a channel from a temperature input board, filters it if specified, does the cold junction compensation, linearization and converts it to temperature.

cbTInScan() - Scan a range of temperature inputs
Reads the temperature from a range of channels as described above. Returns the temperature values to an array.

1.5 COUNTER FUNCTIONS

The counter function names begin with "cbC". These functions load, read and configure counters. There are four types of counter chips used in Computer Board products - 8254s, 8536s, 7266’s and 9513's. Some of the counter commands only apply to one type of counter.

cbC7266Config() - Configures an LS7266 counter
Selects the basic operating mode of an LS7266 counter

cbC8254Config() - Configures 8254 counter
Selects the basic operating mode of an 8254 counter.

cbC8536Config() - Sets operating mode of 8536 counter.
This function sets all of the programmable options that are associated with an 8536 counter chip.

cbC8536Init() - Initializes 8536 counter
Initializes and selects all of the chip level features for a 8536 counter board. The options that are set by this command are associated with each counter chip, not the individual counters within it.

cbC9513Config() - Sets operating mode of 9513 counter.
This function sets all of the programmable options that are associated with a 9513 counter. It is similar in purpose to cbC8254Config() except that it is used with a 9513 counter.

cbC9513Init() - Initializes 9513 counter
Initializes and selects all of the chip level features for a 9513 counter board. The options that are set by this command are associated with each counter chip, not the individual counters within it.

cbCFreqIn() - Measures frequency of a signal
This function measures the frequency of a signal by counting it for a specified period of time (GatingInterval) and then converting the count to count/sec (Hz).   It only works with 9513 counters.

cbCIn() - Reads a counter.
Reads a counter’s current value.

cbCIn32() - Read a counter (currently only applies to LS7266 counters)
Reads a counter’s current value as a 32-bit integer

cbCLoad() - Load a counter.
Loads a counter with an initial count value.

cbCLoad32() -Load a counter (currently only applies to LS7266 counters)
Load’s a counter with an 32-bit integer initial value.

cbCStatus() - Read the counter status (currently only applies to LS7266 counters)
Returns various bits that indicate the current state of a counter

cbCStoreOnInt() - Store counter value when interrupt occurs.
Installs an interrupt handler that will store the current count whenever an interrupt occurs. This function only works with 9513 counters.

1.6 STREAMER FILE FUNCTIONS

The file function names begin with "cbF". These functions create, fill, and read "streamer" files. These functions also let you collect huge amounts of analog input data. The amount of data is limited only by your available disk space.

cbFileAInScan() - Transfer analog input data directly to file.
Very similar to cbAInScan() except that the data is stored in a file instead of an array.

cbFilePretrig() - Pre-triggered analog input to a file.
Very similar to cbAPretrig() except that the data is stored in a file instead of an array.

cbFileGetInfo() - Reads "streamer" file information.
Each streamer file contains information about how much data is in the file and the conditions under which it was collected (sampling rate, channels, etc.). This function reads that information.

cbFileRead() - Reads data from "streamer" file.
Reads a selected number of data points from a streamer file into an array.

1.7 MISC. FUNCTIONS

These functions are the odds and ends. They have to do with error handling and managing background operations.

cbRS485() - Set the transmit and receive buffers on an RS485 port.

cbErrHandling() - Selects type of error handling.
The universal library has a number of different methods of handling errors. This function selects which of these methods will be used with all subsequent library calls. The options include stopping the program when an error occurs and printing error messages.

cbGetErrMsg() - Returns an error message for a given error.
All library functions return error codes. This function converts an error code to an error message.

cbGetStatus() - Returns status of background operation.
Once a background operation is started your program will need to periodically check on its progress. This function returns the current status of the process.

cbStopBackground() - Stop a background process.
It is sometimes necessary to stop a background process even though the process has been set up to run continuously. This function will stop a background process that is running. cbStopBackground() should be executed after normal termination of all background functions in order to clear variables and flags.

cbInByte(), cbInWord() - Reads a byte or word from a hardware register on a board.

cbOutByte(), cbOutWord() - Writes a byte or word to a hardware register on a board.

cbGetConfig() - Returns the current value for a specified configuration option.

cbSetConfig () - Sets the current value for a specified configuration option.

cbSetTrigger () - Sets up trigger parameters used with the EXTTRIGGER option for cbAInScan().

cbGetBoardName() - Returns the name of a specified board.

cbToEngUnits() - Converts a count value from an A/D to voltage (or current).

cbFromEngUnits() - Converts a voltage (or current) to a D/A count value.

1.8 MEMORY BOARD FUNCTIONS

The memory board functions all begin with "cbM". These functions read/write and control memory boards (MEGA-FIFO).

The most common use for the memory boards is to store large amounts of data from an A/D board via a DT-Connect cable between the two boards. To do this, you should use the EXTMEMORY option with cbAInScan() or cbA-Pretrig().

Once the data has been transferred to the memory board you can use the memory functions to retrieve it.

cbMemSetDTMode() - Set DT-Connect Mode on Memory Board
The memory boards have a DT-Connect interface which can be used to transfer data through a cable between two boards rather than through the PC's system memory. The DT-Connect port on the memory board can be configured as either an input (from an A/D) or as an output (to a D/A). This function configures the port.

cbMemReset() - Resets the Memory Board Address
The memory board is organized as a sequential device. When data is transferred to the memory board it is automatically put in the next address location. This function resets the current address to the location 0.

cbMemRead() - Read Data From Memory Board
Reads a specified number of points from a memory board starting at a specified address.

cbMemWrite() - Writes Data To The Memory Board
Writes a specified number of points to a memory board starting at a specified address.

cbMemReadPretrig - Reads Data Collected With cbAPretrig()
The cbAPretrig() function writes the pre-triggered data to the memory board in a scrambled order. This function unscrambles the data and returns it in the correct order.

1.9 WINDOWS MEMORY MANAGEMENT FUNCTIONS

The Windows memory management functions all begin with cbWin. These functions are only available and needed in the Windows version of the library. These functions take care of allocating, freeing and copying to/from

Windows global memory buffers. These functions are not used in VEE since VEE handles memory allocation. For customers wishing to customize memory management under VEE, the source code to CBV.DLL and CBV32.DLL is available. Please call technical support and request it.

cbWinBufAlloc() - Allocate a Windows memory buffer

cbWinBufFree() - Free a Windows buffer

cbWinArrayToBuf() - Copy data from array to Windows buffer

cbWinBufToArray() - Copy data from Windows buffer to array

1.10 REVISION CONTROL FUNCTION

As new revisions of the library are released bugs from previous revisions are fixed and occasionally new functions are added. It is ComputerBoards' goal to preserve existing programs you have written and therefore to never change the order or number of arguments in a function. However, sometimes it is not possible to achieve this goal.

The revision control function initializes the DLL so that the functions are interpreted according to the format of the revision you wrote and compiled your program in. Please see the full explanation of the function provided in the FUNCTIONS manual.

cbDeclareRevision() - Declares the revision # of the Universal Library that your program was written with.

cbGetRevision() - Returns the version number of the installed Universal Library.

2 UNIVERSAL LIBRARY FUNCTIONS

cbDeclareRevision() New R3.3 ID

Description:	Initializes Universal Library with the revision number of the library used to write your program. Must be the first Universal Library function to be called by your program.
Summary:	int cbDeclareRevison(float* RevNum)
Arguments:	RevNum - Revision number of Library to interpret function arguments
Default:	Any program not containing this line of code will be defaulted to revision 3.2 argument assignments.

As new revisions of the library are released bugs from previous revisions are fixed and occasionally new functions are added. It is ComputerBoards goal to preserve existing programs you have written and therefore to never change the order or number of arguments in a function. Sometimes it is not possible to achieve this goal as evidenced in the changes from revision 3.2 to 3.3. In revision 3.3 we added support for multiple background tasks, a feature customers had requested for some time. Allowing multiple background tasks meant adding the argument BoardNum to several functions. Doing so would have meant that programs written for version 3.2 would not run with 3.3 if they called those functions. If not for the new cbDeclareRevision function the programs would have to be rewritten in each line where the affected functions are used, and the program then recompiled.

The revision control function initializes the DLL so that the functions are interpreted according to the format of the revision you wrote and compiled your program in. This function is new in revision 3.3 so to take advantage of it the function must be added to your program and the program recompiled.

The function works by interpreting the UL function call from your program and filling in any arguments needed to run with the new revision. For example, the function cbAConvertData() which appears on the following pages had the argument BoardNum added in revision 3.3. The two revisions of the function look like this:

Rev 3.2	int cbAConvertData (long NumPoints, unsigned ADData[], int ChanTags[])
Rev 3.3	int cbAConvertData (<u>int BoardNum</u>, long NumPoints, unsigned ADData[], int ChanTags[])

If your program has declared you are running code written for revision 3.2 and you call this function, the argument BoardNum is ignored. If you want the benefits afforded by the added argument BoardNum you must rewrite your program with the new argument and declare revision 3.3 (or higher) in cbDeclareRevision().

If a revision less than 3.2 is declared, revision 3.2 is assumed.

Returns: Non-zero if error has occurred.

cbGetRevision()   New R3.3 ID

Description:	Gets the revision level of Universal Library DLL and the VXD.
Summary:	int cbGetRevison(float* DLLRevNum, float* VXDRevNum)
Arguments:	DLLRevNum - Place holder for the revision number of Library DLL VXDRevNum - Place holder for the revision number of Library VXD
Returns:	DLLRevNum - Place holder for the revision number of Library DLL VXDRevNum - Place holder for the revision number of Library VXD Error Code if revision levels of VXD and DLL are incompatible

cbAConvertData()   Changed R3.3 RW

Description: Converts the raw data collected by cbAInScan() into 12 bit A/D values. The cbAInScan() function can return either raw A/D data or converted data depending on whether or not the CONVERTDATA option was used. For many 12 bit A/D boards, the raw data is a 16 bit value that contains a 12 bit A/D value and a 4 bit channel tag (see board specific information or hardware manual). The converted data consists of just the 12 bit A/D value.

Summary: int cbAConvertData (int BoardNum, long NumPoints, unsigned ADData[], int ChanTags[])

Arguments: BoardNum - May be 0 to 99 (0 to 9 for 16 bit version of Universal Library). Refers to the number associated with the board used to collect the data when it was installed with the InstaCal™ configuration program.
NumPoints - Number of samples to convert
ADData - Pointer to data array
ChanTags - pointer to channel tag array

When you collect data with cbAInScan and you don't use the CONVERTDATA option then you may need to use this function to convert the data after it is collected. There are cases where the CONVERTDATA option is not allowed. For example - if you are using both the DMAIO and BACKGROUND option with cbAInScan. In those cases this function should be used to convert the data after the data collection is complete.

On some boards, each raw data point consists of a 12 bit A/D value with a 4 bit channel number. This function pulls each data point apart and puts the A/D value into the ADData array and the channel number into the ChanTags array.

Returns: Error code or 0 if no errors
ADData - converted data
ChanTags - channel tags if available.

Note - 12 Bit A/D Boards
1) Name of array must match that used in cbAInScan.
2) Upon returning from cbAConvertData(), ADData array contains only 12 bit A/D data.

Note - 16 Bit A/D Boards
This function is not for use with 16 bit A/D boards because 16 bit boards do not have channel tags. The argument BoardNum was added in revision 3.3 to prevent applying this function to 16 bit data. If you wrote your program for a 12 bit board then later upgrade to a 16 bit board all you need change is the InstaCal configuration file. If this function is called for a 16 bit board it is simply ignored. No error is generated.

cbAConvertPretrigData()   Changed R3.3 RW

Description: Converts the raw data collected by cbAPretrigger(). The cbAPretrigger() function can return either raw A/D data or converted data depending on whether or not the CONVERTDATA option was used. The raw data as it is collected is not in the correct order. After the data collection is completed it must be rearranged into the correct order. This function correctly orders the data also, starting with the first pretrigger data point and ending with the last post-trigger point.

Change at revision 3.3 is to support multiple background tasks. It is now possible to run 2 boards with DMA or REP-INSW background convert-and-transfer features active, therefore, the convert function must know which board the data came from. The data value assigned to BoardNumber should be assigned in the header file so it will be easy to locate if a change is needed.

Summary: int cbAConvertPretrigData (int BoardNum, long PreTrigCount, long TotalCount, unsigned ADData[], int ChanTags[])

Arguments: BoardNum - May be 0 to 99 (0 to 9 for 16 bit version of Universal Library). The number of the board used to collect the data. Refers to the board number associated with the board when it was installed with the InstaCal™ configuration program.
PreTrigCount - Number of pre-trigger samples (this value must match the value returned by the PreTrigCount argument in the cbPretrig() function)
Total Count - Total number of samples that were collected
ADData - Pointer to data array (must match array name used in cbAPretrig function)
ChanTags - Pointer to channel tag array or a NULL pointer may be passed if using 16-bit boards or if channel tags are not desired (see note regarding 16 bit boards below).

When you collect data with cbAPretrig() and you don't use the CONVERTDATA option then you must use this function to convert the data after it is collected. There are cases where the CONVERTDATA option is not allowed. For example - if you use the BACKGROUND option with cbAPretrig(). In those cases this function should be used to convert the data after the data collection is complete.

Returns: Error code or 0 if no errors
ADData - converted data

Note - 12 Bit A/D Boards
On some 12 bit boards, each raw data point consists of a 12 bit A/D value with a 4 bit channel number. This function pulls each data point apart and puts the A/D value into the ADData and the channel number into the ChanTags array.

Upon returning from cbAConvertPretrigData(), ADData array contains only 12 bit A/D data.

Note - 16 Bit A/D Boards
This function is for use with 16 bit A/D boards only insofar as ordering the data. No channel tags are returned.

Note - Visual Basic Programmers:

After the data is collected with cbAPretrig() it must be copied to a BASIC array with cbWinBufToArray().
Important: The entire array must be copied which includes the extra 512 samples needed by cbAPretrig().

```
Count& = 10000
Dim ADData% (Count& + 512)
Dim ChanTags% (Count& + 512)
cbAPretrig%(BoardNum, LowChan, HighChan, PretrigCount&, Count&...)
cbWinBufToArray%(MemHandle%, ADData%, ), Count& + 512)
cbAConvertPretrigData%(PretrigCount&, Count&, ADData%, ChanTags%)
```

cbACalibrateData() New R3.3

Description: Calibrates the raw data collected by cbAInScan() from boards with real time software calibration when the real time calibration has been turned off. The cbAInScan() function can return either raw A/D data or calibrated data depending on whether or not the NOCALIBRATEDATA option was used.

Summary: int cbACalibrateData (int BoardNum, long NumPoints, int Gain, unsigned ADData[])

Arguments: BoardNum - May be 0 to 99 (0 to 9 for 16 bit version of Universal Library). Refers to the board number associated with the board when it was installed with the InstaCal™ configuration program.
NumPoints - Number of samples to convert
Gain - The programmable gain/range used when the data was collected
ADData - Pointer to data array

When you collect data with cbAInScan and you use the NOCALIBRATEDATA option then you must use this function to calibrate the data after it is collected.

Returns: Error code or 0 if no errors
ADData - converted data

Note -
1) Name of array must match that used in cbAInScan().

cbAIn()

Description: Reads an A/D input channel. This function reads the specified A/D channel from the specified board. If the specified A/D board has programmable gain then it sets the gain to the specified range. The raw A/D value is converted to an A/D value and returned to DataValue.

Summary: int cbAIn (int BoardNum, int Channel, int Range, unsigned *DataValue);

Arguments: BoardNum - May be 0 to 99 (0 to 9 for 16 bit version of Universal Library).
Channel - A/D channel number
Range - A/D Range code
DataValue - Pointer to data value

Returns: Error code or 0 if no errors
DataValue - Value of A/D sample returned here

BoardNum - refers to the board number associated with the board when it was installed with the InstaCal™ configuration program. The specified board must have an A/D.

Channel - The maximum allowable channel depends on which type of A/D board is being used. For boards that have both single ended and differential inputs the maximum allowable channel number also depends on how the board is configured. For example, a CIO-DAS1600 has 8 chans for differential, 16 for single ended. Expansion boards are also supported by this function so this argument could contain values as high as 256. See the board specific information for EXP boards if you are using an expansion board.

Range - If the selected A/D board does not have a programmable gain feature then this argument will be ignored. If the A/D board does have programmable gain then set the Range argument to the desired A/D range. Not all A/D boards support the same A/D ranges. Refer to the A/D board manual for a list of supported A/D Ranges.

BIP10VOLTS	+/- 10 volts	UNI10VOLTS	0 - 10 volts
BIP5VOLTS	+/- 5 volts	UNI5VOLTS	0 - 5 volts
BIP2PT5VOLTS	+/- 2.5 volts	UNI2PT5VOLTS	0 - 2.5 volts
BIP1PT25VOLTS	+/- 1.25 volts	UNI2VOLTS	0 - 2 volts
BIP1VOLTS	+/- 1 volts	UNI1PT25VOLTS	0 - 1.25 volts
BIP1PT67VOLTS	+- 1.67 volts	UNI1PT67VOLTS	0 - 1.67 volts
BIPPT625VOLTS	+/- 0.625 volts	UNI1VOLTS	0 - 1 volts
BIPPT5VOLTS	+/- 0.5 volts	UNIPT1VOLTS	0 - 0.1 volts
BIPPT1VOLTS	+/- 0.1 volts	UNIPT01VOLTS	0 - 0.01 volts
BIPPT05VOLTS	+/-0.05 volts	MA4TO20	4 - 20 mA
BIPPT01VOLTS	+/- 001 volts	MA2TO10	2 - 10 mA
BIPPT005VOLTS	+/- 0.005 volts	MA1TO5	1 - 5 mA
		MAPT5TO2PT5	0.5 - 2.5 mA

cbAInScan() Changed R3.3 ID

Description: Scans a range of A/D channels and stores the samples in an array. This function reads the specified number of A/D samples at the specified sampling rate from the specified range of A/D channels from the specified board. If the A/D board has programmable gain then it sets the gain to the specified range. The collected data is returned to the data array.

Changes: Revision 3.3 added ‘no real time calibration’ option.

Summary: DOS Language:
int cbAInScan (int BoardNum, int LowChan, int HighChan, long Count, long *Rate, int Range, unsigned ADData[], int Options)
Windows Language:
int cbAInScan (int BoardNum, int LowChan, int HighChan, long Count, long *Rate, int Range, int MemHandle, int Options)

Arguments: BoardNum - May be 0 to 99 (0 to 9 for 16 bit version of Universal Library).
LowChan - First A/D channel of scan
HighChan - Last A/D channel of scan
Count - Number of A/D samples to collect
Rate - Sample rate in scans per second per channel
Range - A/D range code
ADData - Data array to store A/D values in (DOS)
MemHandle - Handle for Windows buffer to store data in (Windows)
In VEE this panel is called Data Array. See VEE specific information for more details
Options - Bit fields that control various options

EXPLANATION OF THE ARGUMENTS:

BoardNum - refers to the board number associated with the board when it was installed with the InstaCal configuration program. The specified board must have an A/D.

Low / High Channel # - Indicates the range of channels to scan. LowChan must be less than or equal to HighChan. The maximum allowable channel depends on which type of A/D board is being used. For boards that have both single ended and differential inputs the maximum allowable channel number also depends on how the board is configured. For example, a CIO-DAS1600 has 8 chans for differential, 16 for single ended.

Count - Specifies the total number of A/D samples that will be collected. If more than one channel is being sampled then the number of samples collected per channel is equal to Count / (HighChan- LowChan+1).

Rate - This is the rate at which scans are triggered. If you are sampling 4 channels, 0-3, then specifying a rate of 10,000 scans per second (10KHz) will result in the A/D converter rate of 40KHz: 4 channels at 10,000 samples per channel per second. This is different from some software where you specify the total A/D chip rate. In those systems, the per channel rate is equal to the A/D rate divided by the number of channels in a scan. This argument also returns the value of the actual rate set. This may be different from the requested rate because of pacer limitations.

Caution! You will generate an error if you specify a total A/D rate beyond the capability of the board. For example; if you specify rate LowChan = 0, HighChan = 7 (8 channels total) and Rate = 20,000 and you are using a CIO-DAS16/Jr, you will get an error. You have specified a total rate of 8*20,000 = 160,000. The CIO-DAS16/Jr is capable of converting 120,000 samples per second. The maximum sampling rate depends on the A/D board that is being used. It is also dependent on the sampling mode options.

Range - If the selected A/D board does not have a programmable range feature, then this argument will be ignored. Otherwise the gain can be set to any of the following ranges that are supported by the selected A/D board. Refer to board specific information for the list of ranges supported by each board.

BIP10VOLTS	+/- 10 volts	UNI10VOLTS	0 - 10 volts
BIP5VOLTS	+/- 5 volts	UNI5VOLTS	0 - 5 volts
BIP2PT5VOLTS	+/- 2.5 volts	UNI2PT5VOLTS	0 - 2.5 volts
BIP1PT25VOLTS	+/- 1.25 volts	UNI2VOLTS	0 - 2 volts
BIP1PT67VOLTS	+/- 1.67 volts	UNI1PT67VOLTS	0 - 1.67 volts
BIP1VOLTS	+/- 1 volts	UNI1PT25VOLTS	0 - 1.25 volts
BIPPT625VOLTS	+/- 0.625 volts	UNI1VOLTS	0 - 1 volts
BIPPT5VOLTS	+/- 0.5 volts	UNIPT1VOLTS	0 - 0.1 volts
BIPPT1VOLTS	+/- 0.1 volts	UNIPT01VOLTS	0 - 0.01 volts
BIPPT05VOLTS	+/-0.05 volts	MA4TO20	4 - 20 mA
BIPPT01VOLTS	+/- 001 volts	MA2TO10	2 - 10 mA
BIPPT005VOLTS	+/- 0.005 volts	MA1TO5	1 - 5 mA
		MAPT5TO2PT5	0.5 - 2.5 mA

ADData - The data array must be big enough to hold at least Count number of integers.

MemHandle - Handle for Windows buffer to store data in (Windows). This buffer must have been previously allocated with the cbWinBufAlloc() function.

Options - This field may contain any combination of non-contradictory choices from the following list:

DTCONNECT - All A/D values will be sent to the A/D board's DT CONNECT port. This option is incorporated into the EXTMEMORY option. Use DTCONNECT only if the external board is not supported by Universal Library.

Trigger & Transfer Method Options: If none of these options is specified (recommended), the optimum sampling mode will automatically be chosen based on board type and sampling speed.

> SINGLEIO - A/D conversions and transfers to memory are initiated by an interrupt. One interrupt per conversion.
>
> DMAIO - A/D conversions are initiated by a trigger. Transfers are initiated by a DMA request.
>
> BLOCKIO - A/D conversions are initiated by a trigger. Transfers are handled in blocks (by REPINSW for example). If the rate of acquisition is very slow (say less than 200 Hz) BLOCKIO is probably not the best choice for transfer mode. The reason for this is that status for the operation is not available until one packets worth of data has been collected (typically 512 samples). The

implication is that, if acquiring 100 samples at 100 Hz using BLOCKIO, the operation will not complete until 5.12 seconds has elapsed.

BURSTMODE - Enables burst mode sampling. Scans from LowChan to HighChan are clocked at the maximum A/D rate between samples in order to minimize channel to channel skew. Scans are initiated at the rate specified by the Rate argument.

CONVERTDATA - If the CONVERTDATA option is used for 12 bit boards then the data that is returned to ADData will automatically be converted to 12 bit A/D values. If CONVERTDATA is not used then the data from 12 bit A/D boards will be return unmodified (16 bit values that contain both a 12 bit A/D value and a 4 bit channel number). After the data collection is complete you can call cbAConvertData() to convert the data after the fact. CONVERTDATA may not be specified if you are using the BACKGROUND option and DMA transfers. This option is ignored for the 16 bit boards.

BACKGROUND - If the BACKGROUND option is not used then the cbAInScan() function will not return to your program until all of the requested data has been collected and returned to ADData. When the BACKGROUND option is used, control will return immediately to the next line in your program and the data collection from the A/D into ADData will continue in the background. Use cbGetStatus() to check on the status of the background operation. Use cbStopBackground() to terminate the background process before it has completed. cbStopBackground() should be executed after normal termination of all background functions in order to clear variables and flags.

CONTINUOUS - This option puts the function in an endless loop. Once it collects the required number of samples, it resets to the start of ADData and begins again. The only way to stop this operation is with cbStopBackground(). Normally this option should be used in combination with BACKGROUND so that your program will regain control.

NOTE: In some cases, the minimum value for the Count argument may change when the CONTINUOUS option is used. This can occur for several reasons. The most common is that in order to trigger an interrupt on boards with FIFOs, the circular buffer must occupy at least half the FIFO. Typical half - FIFO sizes are 256, 512 and 1024. See the board specific information to determine packet size for the board you are using.

Another reason for a minimum Count value is that the buffer in memory must be periodically transferred to the user buffer. If the buffer is too small, data will be overwritten during the transfer resulting in garbled data.

EXTCLOCK - If this option is used then conversions will be controlled by the signal on the trigger input line rather than by the internal pacer clock. Each conversion will be triggered on the appropriate edge of the trigger input signal (see board-specific information). When this option is used the Rate argument is ignored. The sampling rate is dependent on the trigger signal. Options for the board will default to a transfer mode that will allow the maximum conversion rate to be attained unless otherwise specified.

NOTE: If the rate of the external clock is very slow (say less than 200 Hz) and the board you are using supports BLOCKIO, you may want to include the SINGLEIO option. The reason for this is that status for the operation is not available until one packets worth of data has been collected (typically 512 samples). The implication is that, if acquiring 100 samples at 100 Hz using BLOCKIO (the default for boards that support it if EXTCLOCK is used), the operation will not complete until 5.12 seconds has elapsed.

EXTMEMORY causes the command to send the data to a connected memory board via the DT-Connect interface rather than returning the data to ADData. Data for each call to this function will be appended unless cbMemReset is called. The data should be unloaded with the cbMemRead() function before collecting new data. When EXTMEMORY option is used, the ADData() array can be set to null or 0. CONTINUOUS option cannot be used with EXTMEMORY. Do not use EXTMEMORY with DTCONNECT, SINGLEIO, DMAIO or BLOCKIO.

EXTTRIGGER - If this option is specified the sampling will not begin until the trigger condition is met. On many boards, this trigger condition is programmable (see cbSetTrigger() function and board-specific information for details). On other boards, only 'polled gate' triggering is supported. In this case assuming active high operation, data acquisition will commence immediately if the trigger input is high. If the trigger input is low, acquisition will be held off until it goes high. Acquisition will then continue until NumPoints & samples have been taken regardless of the state of the trigger input. This option is most useful if the signal is a pulse with a very low duty cycle (trigger signal in TTL low state most of the time) so that triggering will be held off until the occurrence of the pulse.

NOTODINTS - If this option is specified then the system's time-of-day interrupts are disabled for the duration of the scan. These interrupts are used to update the systems real time clock and are also used by various other programs. These interrupts can limit the maximum sampling speed of some boards - particularly the PCM-DAS08. If the interrupts are turned off using this option then the realtime clock will fall behind by the length of time that the scan takes.

NOCALIBRATEDATA - Turns off real time software calibration for boards which are software calibrated by applying calibration factors to the data on a sample by sample basis as it is acquired. Examples are the PCM-DAS16/330 and PCM-DAS16x/12. Turning off software calibration saves CPU time during a high speed acquisition run. This may be required if your processor is less than a 150MHz Pentium and you desire an acquisition speed in excess of 200KHz. These numbers may not apply to your system. Only trial will tell for sure. DO NOT use this option if you do not have to. If this option is used the data must be calibrated after the acquisition run with the cbACalibrateData() function.

RETURNS:	error code or 0 if no errors
	Rate = actual sampling rate used
	ADData - Collected A/D data returned here

VERY IMPORTANT NOTE

In order to understand the functions, you must read the <u>Board Specific Information</u> section found in the UL User's Guide manual. The example programs should be examined and run prior to attempting any programming of your own. Following this advice will save you hours of frustration, and possibly time wasted holding for technical support.

This note, which appears elsewhere, is especially applicable to this function. Now is the time to read the board specific information for your board (see Universal Library User's Guide). We suggest that you make a copy of that page to refer to as you read this manual and examine the example programs.

cbALoadQueue()

Description: Loads A/D board's channel/gain queue. This function only works with A/D boards that have channel/gain queue hardware.

Summary: int cbALoadQueue (int BoardNum, int ChanArray[], int GainArray[], int Count);

Arguments:	BoardNum - May be 0 to 99 (0 to 9 for 16 bit version of Universal Library).
	ChanArray - Array containing channel values
	GainArray - Array containing A/D range values
	Count - Number of elements in ChanArray and GainArray or 0 to disable chan/gain queue

EXPLANATION OF THE ARGUMENTS:

BoardNum - refers to the board number associated with the board when it was installed with the configuration program. The specified board must have an A/D and a channel/gain queue.

ChanArray - This array should contain all of the channels that will be loaded into the channel gain queue.

GainArray - This array should contain each of the A/D ranges that will be loaded into the channel gain queue.

Count - Specifies the total number of chan/gain pairs that will be loaded into the queue. ChanArray and GainArray should contain at least Count elements. Set Count=0 to disable the board's chan/gain queue. The maximum value is specific to the queue size of the A/D boards channel gain queue.

Normally the cbAInScan() function scans a fixed range of channels (from LowChan to HighChan) at a fixed A/D range. If you load the channel gain queue with this function then all subsequent calls to cbAInScan() will cycle through the chan/range pairs that you have loaded into the queue.

Returns: error code or 0 if no errors

cbAOut()

Description: Sets the value of a D/A output

Summary: int cbAOut (int BoardNum, int Channel, int Range, unsigned DataValue)

Arguments: BoardNum - May be 0 to 99 (0 to 9 for 16 bit version of Universal Library).
Channel - A/D channel number
Range - D/A range code
DataValue - Value to set D/A to

EXPLANATION OF THE ARGUMENTS:

BoardNum - refers to the board number associated with the board when it was installed with the configuration program.   The specified board must have a D/A.

Channel - The maximum allowable channel depends on which type of D/A board is being used.

Range - The output range of the D/A channel can be set to any of those supported by the board. If the D/A board does not have programmable ranges then this argument will be ignored.

DataValue - Must be in the range 0 - N where N is the value 2 ^ Resolution - 1 of the converter (Exception: using 16 bit boards with Basic range is -32768 to 32767. See notes on signed integers and basic elsewhere in this manual).

Returns: Error code or 0 if no errors

Simultaneous Update Boards: If you have set the simultaneous update jumper for simultaneous operation, you should use cbAOutScan() for simultaneous update of multiple channels. cbAOut() always writes the D/A data then reads the D/A, which causes the D/A output to be updated.

cbAOutScan()

Description:	Outputs values to a range of D/A channels

Summary: DOS Language:
int cbAOutScan (int BoardNum, int LowChan, int HighChan, long Count, long *Rate, int Range, unsigned DAData[], int Options)
Windows Language:
int cbAOutScan (int BoardNum, int LowChan, int HighChan, long Count, long *Rate, int Range, int MemHandle, int Options)

Arguments:
BoardNum - May be 0 to 99 (0 to 9 for 16 bit version of Universal Library).
LowChan - First D/A channel of scan
HighChan - Last D/A channel of scan
Count - Number of D/A values to output
Rate - Sample rate in scans per second
Range -D/A range code or 0
DAData - Data array to store D/A values in (DOS)
MemHandle - Handle for Windows buffer from which data will be output
In VEE this panel is called Data Array. See VEE specific information for more details
Options - Bit fields that control various options

EXPLANATION OF THE ARGUMENTS:

BoardNum - refers to the board number associated with the board when it was installed with the configuration program. The specified board must have a D/A.

LowChan/HighChan - The maximum allowable channel depends on which type of D/A board is being used.

Count - Specifies the total number of D/A values that will be output. Most D/A boards do not support timed outputs. For these boards count should be set to the number of channels in the scan.

Rate - For many D/A boards the Rate is ignored and can be set to NOTUSED. For D/A boards with trigger and transfer methods which allow fast output rates, such as the CIO-DAC04/12-HS, Rate should be set to the D/A output rate (in scans/sec). This argument will return the actual value of the rate set. This may be different from the requested rate because of pacer limitations.

If supported, this is the rate at which scans are triggered. If you are updating 4 channels, 0-3, then specifying a rate of 10,000 scans per second (10KHz) will result in the D/A converter rates of 10KHz: (one D/A per channel). The data transfer rate will be 40,000 words per second; 4 channels * 10,000 updates per scan.
Caution! You will generate an error if you specify a total D/A rate beyond the capability of the board. For example; if you specify rate LowChan = 0, HighChan = 3 (4 channels total) and Rate = 100,000 and you are using a cSBX-DDA04, you will get an error. You have specified a total rate of 4*100,000 = 400,000. The cSBX-DDA04 is rated to 330,000 updates per second.

The maximum update rate depends on the D/A board that is being used. It is also dependent on the sampling mode options.

Range - The output range of the D/A channel can be set to any of those supported by the board. If the D/A board does not have a programmable then this argument will be ignored.

DAData - The data array should be filled with D/A values in the range 0 - N where N is the value 2 ^ Resolution-1 of the converter. There should be at least HighChan-LowChan+1 elements in the array. (Exception: using 16 bit boards with Basic - range is -32768 to 32767. See notes on signed integers and basic elsewhere in this manual).

MemHandle - Handle for Windows buffer from which data will be output. This buffer must have been previously allocated with the cbWinBufAlloc() function and data values loaded (perhaps using cbwin in Array to Buf()).

Options:

CONTINUOUS - This option may only be used with boards which support interrupt, DMA or REP-INSW transfer methods.   This option puts the function in an endless loop. Once it outputs the specified (by Count) number of D/A values, it resets to the start of DAData and begins again. The only way to stop this operation is with cbStopBackground(). This option should only be used in combination with BACKGROUND so that your program can regain control.

BACKGROUND - This option may only be used with boards which support interrupt, DMA or REP-INSW transfer methods. When this option is used the D/A operations will begin running in the background and control will immediately return to the next line of your program. Use cbGetStatus() to check the status of background operation. Use cbStopBackground() to terminate background operations before they are completed.

SIMULTANEOUS - When this option is used (if the board supports it and the appropriate switches are set on the board) all of the D/A voltages will be updated simultaneously when the last D/A in the scan is updated. This generally means that all the D/A values will be written to the board, then a read of a D/A address causes all D/As to be updated with new values simultaneously.

EXTCLOCK - If this option is used then conversions will be paced by the signal on the trigger input line rather than by the internal pacer clock.   Each conversion will be triggered on the appropriate edge of the trigger input signal (see board-specific information).   When this option is used the Rate argument is ignored. The sampling rate is dependent on the trigger signal. Options for the board will default to transfer types that allow the maximum conversion rate to be attained unless otherwise specified.

EXTTRIGGER - If this option is specified the sampling will not begin until the trigger condition is met. On many boards, this trigger condition is programmable (see cbSetTrigger() function and board-specific information for details).

Returns: Error code or 0 if no errors

cbAPretrig()

Description: Waits for a trigger to occur and then returns a specified number of analog samples before and after the trigger occurred. If only 'polled gate' triggering is supported, the trigger input line (see board user's manual) must be at TTL low before this function is called or a TRIGSTATE error will occur. The trigger occurs when the trigger condition is met. See cbSetTrigger() function and board-specific information for details.

Summary: DOS Language:

```
int cbAPretrig (int BoardNum, int LowChan, int HighChan, long *PretrigCount,
long *TotalCount, long *Rate, int Range, unsigned ADData[],  int Options )
```

Windows Language:

```
int cbAPretrig (int BoardNum, int LowChan, int HighChan, long *PretrigCount,
long *TotalCount, long *Rate, int Range, int MemHandle,  int Options )
```

Arguments:	BoardNum - May be 0 to 99 (0 to 9 for 16 bit version of Universal Library).
	LowChan - First A/D channel of scan
	HighChan - Last A/D channel of scan
	PretrigCount - Number of pre-trigger A/D samples to collect
	TotalCount - Total number of A/D samples to collect.
	Rate - Sample rate in scans per second
	Range - A/D Range code or 0
	ADData - Data array to store A/D values in. (DOS)
	MemHandle - Handle for Windows buffer to store data in (Windows)
	IMPORTANT: the data array (or Windows buffer) must be big enough to hold TotalCount+512 integers
	Options - Bit fields that control various options

EXPLANATION OF THE ARGUMENTS:

BoardNum - refers to the board number associated with the board when it was installed with the configuration program. The specified board must have an A/D.

LowChan/HighChan - The maximum allowable channel depends on which type of A/D board is being used. For boards that have both single ended and differential inputs the maximum allowable channel number also depends on how the board is configured (8 chans for differential, 16 for single ended).

PretrigCount - Specifies the number of samples before the trigger that will be returned. PretrigCount must be less than 32000 and PretrigCount must also be less than TotalCount-512. If the trigger occurs too early, then fewer than the requested number of pre-trigger samples will be collected. In that case a TOOFEW error will occur. The PretrigCount will be set to indicate how many samples were collected and the post trigger samples will still be collected.

TotalCount - Specifies the total number of samples that will be collected and stored in ADData. TotalCount must be greater than or equal to PretrigCount+512. If the trigger occurs too early then fewer than the requested number of samples will be collected. In that case a TOOFEW error will occur. The TotalCount will be set to indicate how many samples were actually collected. Also, TotalCount should be evenly divisible by the number of channels being scanned. If it is not, this function will adjust the number down to the next valid value and return that value to the TotalCount argument.

Range - If the selected A/D board does not have a programmable gain feature then this argument is ignored. Otherwise the Range can be set to any of the following ranges that are supported by the selected A/D board. Refer to board specific information for a list of the A/D ranges supported by each board.

BIP10VOLTS	+/- 10 volts	UNI10VOLTS	0 - 10 volts
BIP5VOLTS	+/- 5 volts	UNI5VOLTS	0 - 5 volts
BIP2PT5VOLTS	+/- 2.5 volts	UNI2PT5VOLTS	0 - 2.5 volts
BIP1PT25VOLTS	+/- 1.25 volts	UNI2VOLTS	0 - 2 volts
BIP1PT67VOLTS	+/- 1.67 volts	UNI1PT67VOLTS	0-1.67 volts
BIP1VOLTS	+/- 1 volts	UNI1PT25VOLTS	0 - 1.25 volts
BIPPT625VOLTS	+/- 0.625 volts	UNI1VOLTS	0 - 1 volts
BIPPT5VOLTS	+/- 0.5 volts	UNIPT1VOLTS	0 - 0.1 volts
BIPPT1VOLTS	+/- 0.1 volts	UNIPT01VOLTS	0 - 0.01 volts
BIPPT05VOLTS	+/-0.05 volts	MA4TO20	4 - 20 mA
BIPPT01VOLTS	+/- 001 volts	MA2TO10	2 - 10 mA
BIPPT005VOLTS	+/- 0.005 volts	MA1TO5	1 - 5 mA
		MAPT5TO2PT5	0.5 - 2.5 mA

ADData - The data array for the pretrigger data.

MemHandle - Handle for Windows buffer to store data in (Windows). This buffer must have been previously allocated with the cbWinBufAlloc() function.

VERY IMPORTANT: The ADData or MemHandle data array must be big enough to hold at least TotalCount+512 integers

Options:

DTCONNECT - When DTCONNECT option is used with this function the data from ALL A/D conversions is sent out the DT-CONNECT interface. While this function is waiting for a trigger to occur, it will send data out the DT-CONNECT interface continuously. If you have a Computer Boards memory board plugged into the DT-CONNECT interface then you should use EXTMEMORY option rather than this option.

EXTMEMORY - If you use this option to send the data to a MEGA-FIFO memory board then you must use cbMemReadPretrig() to later read the pre-trigger data from the memory board. If you use cbMemRead() the data will NOT be in the correct order. Every time this option is used it will overwrite any data that is already stored in the memory board. All data should be read from the board (with cbMemReadPretrig()) before collecting any new data. When this option is used the ADData argument is ignored. The MEGA-FIFO memory must be fully populated in order to use the cbAPretrig() function with the EXTMEMORY option.

CONVERTDATA - The data is collected into a "circular" buffer. When the data collection is complete the data is in the wrong order. If you use the CONVERTDATA option then the data will be automatically rotated into the correct order (and converted to 12 bit values if required) when the data acquisition is complete. Otherwise you must call cbAConvertPretrigData() to rotate the data. You can not use the CONVERTDATA option in combination with the BACKGROUND option.

BACKGROUND - If the BACKGROUND option is not used then the cbAPretrig() function will not return to your program until all of the requested data has been collected and returned to ADData. When the BACKGROUND option is used, control will return immediately to the next line in your program and the data collection from the A/D into ADData will continue in the background. Use cbGetStatus() to check on the status of the background operation. Use cbStopBackground() to terminate the background process before it has completed. cbStopBackground() should also be called after normal termination of all background functions in order to clear variables and flags. When using background, you cannot use the CONVERTDATA option. To correctly order and parse the data, use cbAConvertPretrigData().

EXTCLOCK - This option is available only for boards that have separate inputs for external pacer and external trigger. See your hardware manual or board specific information.

Returns: error code or 0 if no errors
PretrigCount - Number of pre-trigger samples
TotalCount - Total number of samples collected
Rate = actual sampling rate
ADData - Collected A/D data returned here

cbATrig()

Description: Waits for a specified analog input channel to go above or below a specified value. This function continuously reads the specified channel and compares its value to TrigValue. Depending on whether TrigType is ABOVE or BELOW it waits for the first A/D sample that is above or below TrigValue. It returns the first sample that meets the trigger criteria to DataValue.

Summary: int cbATrig (int BoardNum, int Chan, int TrigType, int TrigValue, int Range, unsigned *Data-Value)

Arguments: BoardNum - May be 0 to 99 (0 to 9 for 16 bit version of Universal Library).
Chan - A/D channel number
TrigType - TRIGABOVE or TRIGBELOW - Specifies whether waiting for the analog input to be ABOVE or BELOW the specified trigger value.
TrigValue - The threshold value that all A/D values are compared to.
Range - Gain code
DataValue - The value of the first A/D sample that met the trigger criteria is returned here.

EXPLANATION OF THE ARGUMENTS:

BoardNum - refers to the board number associated with the board when it was installed with the configuration program. The specified board must have an A/D.

Chan - The maximum allowable channel depends on which type of A/D board is being used. For boards that have both single ended and differential inputs the maximum allowable channel number also depends on how the board is configured. For example a CIO-DAS1600 has 8 chans for differential, 16 for single ended.

TrigValue - Must be in the range 0 - 4095 for 12 bit A/D boards, or 0-65,535 for 16 bit A/D boards. *Please see notes on BASIC integer data types in your BASIC manual.*

Range - If the selected A/D board does not have a programmable gain feature then this argument will be ignored. Otherwise the Range can be set to any of the following ranges that are supported by the selected A/D board. Refer to board specific information for a list of the A/D ranges supported by each board.

BIP10VOLTS	+/- 10 volts	UNI10VOLTS	0 - 10 volts
BIP5VOLTS	+/- 5 volts	UNI5VOLTS	0 - 5 volts
BIP2PT5VOLTS	+/- 2.5 volts	UNI2PT5VOLTS	0 - 2.5 volts
BIP1PT67VOLTS	+/- 1.67 volts	UNI1PT67VOLTS	0 - 1.67 volts
BIP1PT25VOLTS	+/- 1.25 volts	UNI2VOLTS	0 - 2 volts
BIP1VOLTS	+/- 1 volts	UNI1PT25VOLTS	0 - 1.25 volts
BIPPT625VOLTS	+/- 0.625 volts	UNI1VOLTS	0 - 1 volts
BIPPT5VOLTS	+/- 0.5 volts	UNIPT1VOLTS	0 - 0.1 volts
BIPPT1VOLTS	+/- 0.1 volts	UNIPT01VOLTS	0 - 0.01 volts
BIPPT05VOLTS	+/-0.05 volts	MA4TO20	4 - 20 mA
BIPPT01VOLTS	+/- 001 volts	MA2TO10	2 - 10 mA
BIPPT005VOLTS	+/- 0.005 volts	MA1TO5	1 - 5 mA
		MAPT5TO2PT5	0.5 - 2.5 mA

Returns: - Error code or 0 if no errors

CTRL-C will not terminate the wait for an analog trigger that meets the specified condition. The only two ways to terminate this call are to satisfy it or reset the computer.

Windows Caution - This function should be used with caution in Windows programs. All active windows will be locked on the screen until the trigger condition is satisfied. All keyboard and mouse will also be locked until the trigger condition is satisfied.

2.1 COUNTERS - AN INTRODUCTION

Universal Library provides functions for initialization and configuration of counter chips. It is important to note what this means:

1. Universal Library can configure a counter for any of the counter operations.

2. Counter configuration does not include USE of counters such as event counting and pulse width. Counter use is accomplished by programs which use the counter functions.

3. Some counter USE functions may be available. A function cbFreqIn() is provided (Revision 1 on). Others may be added to later revisions.

For you to use a counter for any but the simplest counting function, you must read, understand and employ the information contained in the chip manufacturer's data sheet. Technical support of the Universal Library does not include providing, interpreting or explaining the counter chip data sheet.

Counter chip data sheets are available from:

82C54
Intel Corporation
http://www.intel.com/design/litcentr/litweb/suprh.htm

AM9513
Call ComputerBoards Tech Support
(508) 946-9500

Z8536
As of this writing, the only ComputerBoards part that employs the Z8536 is the CIO-INT32. The data book for the chip is included with the CIO-INT32.

LS7266
US Digital
http://www.usdigital.com

Counter Chip Variables
Universal Library counter initialization and configuration functions include names for bit patterns, such as ALEGATE, which stands for Active Low Enabled Gate N. In any case where Universal Library has a name for a bit pattern, it is allowed to substitute the bit pattern as a numeric. This will work, but your programs will be harder to read and debug.

cbC7266Config()

Description: Configures 7266 counter for desired operation. This function can only be used with boards that contain a 7266 counter chip (Quadrature Encoder boards).

Summary: int cbC7266Config (int BoardNum, int CounterNum, int Quadrature, int CountingMode, int DataEncoding, int IndexMode, int InvertIndex, int FlagPins, int Gating);

Arguments: BoardNum - May be 0 to 99 (0 to 9 for 16 bit version of Universal Library).
CounterNum - Counter number (1 - n) to configure
Quadrature - NO_QUAD, X1_QUAD, X2_QAUD or X4_QUAD
CountingMode - NORMAL_MODE, RANGE_LIMIT, NO_RECYCLE, MODULO_N
DataEncoding - BCD_ENCODING, BINARY_ENCODING
IndexMode - INDEX_DISABLED, LOAD_CTR, LOAD_OUT_LATCH, RESET_CTR
InvertIndex - DISABLED or ENABLED
FlagPins - Selects function for X1FLG and X2FLG pins. CARRY_BORROW, COMPARE_BORROW, CARRYBORROW_UPDOWN, INDEX_ERROR.
Gating - DISABLED or ENABLED

EXPLANATION OF THE ARGUMENTS:

BOARDNUM **-** Refers to the board number associated with the board when it was installed with the configuration program. The specified board must have an LS7266 counter.

COUNTERNUM **-** Counter Number (1 - n) where n is the number of counters on the board. A PCM-QUAD02 or CIO-QUAD02 has two counters. A CIO-QUAD04 has four counters.

QUADRATURE - Selects the resolution multiplier (X1_QUAD, X2_QUAD, or X4_QUAD) for quadrature input or disables quadrature input (NO_QUAD) so that the counters can be used as standard TTL counters.

COUNTINGMODE - Selects operating mode for the counter.

NORMAL_MODE - Each counter operates as a 24 bit counter that rolls over to 0 when the maximum count is reached.

RANGE_LIMIT - In range limit count mode, an upper an lower limit is set, mimicking limit switches in the mechanical counterpart. The upper limit is set by loading the PRESET register with the cbCLoad function after the counter has been configured.. The lower limit is always 0. When counting up, the counter freezes whenever the count reaches the value that was loaded into the PRESET register. When counting down, the counter freezes at 0. In either case the counting is resumed only when the count direction is reversed.

NO_RECYCLE - In non-recyle mode the counter is disabled whenever a count overflow or underflow takes place. The counter is re-enabled when a reset or load operation is performed on the counter.

MODULO_N - In modulo-n mode an upper limit is set by loading the PRESET register with a maximum count. Whenever counting up, when the maximum count is reached, the counter will roll-over to 0 and continue counting up.   Likewise when counting down, whenever the count reaches 0, it will roll over to the maximum count (in the PRESET register) and continue counting down.

DATAENCODING - Selects the format of the data that is returned by the counter - either Binary or BCD format.

INDEXMODE - Selects which action will be taken when the Index signal is received. The IndexMode must be set to INDEX_DISABLED whenever a Quadrature is set to NON_QUAD or when Gate is set to ENABLED.

INDEX_DISABLED - The Index signal is ignored

LOAD_CTR - The counter is loaded whenever the Index signal ON the LCNTR pin occurs

LOAD_OUT_LATCH - The current count is latched whenever the Index signal on the LCNTR pin occurs. When this mode is selected, The CIn() function will return the same count each time it is called until the Index signal occurs.

RESET_CTR - The counter is reset to 0 whenever the Index signal on the RCNTR pin occurs

INVERTINDEX - Selects the polarity of the Index signal. If set to DISABLED the Index signal is assumed to be positive polarity. If set to ENABLED the Index signal is assumed to be negative polarity.

FLAGPINS **-** Selects which signals will be routed to the FLG1 and FLG2 pins.
CARRY_BORROW - FLG1 pin is CARRY output, FLG2 is BORROW output

COMPARE_BORROW - FLG1 pin is COMPARE output, FLG2 is BORROW output

CARRYBORROW_UPDOWN - FLG1 pin is CARRY/BORROW output, FLG2 is UP/DOWN signal

INDEX_ERROR - FLG1 is INDEX output, FLG2 is error output

GATING - If gating is set to ENABLED then the RCNTR pin will be used as a gating signal for the counter.   Whenever Gating=ENABLED the IndexMode must be set to DISABLE_INDEX.

Returns: Error code or 0 if no error occurs

cbC8254Config()

Description: Configures 8254 counter for desired operation. This function can only be used with 8254 counters.

Summary: int cbC8254Config (int BoardNum, int CounterNum, int Config)

Arguments: BoardNum - May be 0 to 99 (0 to 9 for 16 bit version of Universal Library).
CounterNum - counter number to configure
Config - the action to take on terminal count and the wave form if any.

EXPLANATION OF THE ARGUMENTS:
BoardNum - refers to the number associated with the board when it was installed with the configuration program.

CounterNum - Selects one of the counter channels. An 8254 has 3 counters. The value may be 1 - n, where n is the number of 8254 counters on the board (see board-specific information).

Config - Refer to the 8254 data sheet for a detailed description of each of the configurations. May be set to one of the following constants:

HIGHONLASTCOUNT:	Output of counter (OUT N) transitions from low to high on terminal count and remains high until reset. See Mode 0 on 8254 data sheet.
ONESHOT:	Output of counter (OUT N) transitions from high to low on rising edge of GATE N, then back to high on terminal count. See mode 1 on 8254 data sheet.
RATEGENERATOR:	Output of counter (OUT N) pulses low for one clock cycle on terminal count and reloads the counter and recycles. See mode 2 on 8254 data sheet.
SQUAREWAVE :	Output of counter (OUT N) is high for count < 1/2 terminal count then low until terminal count, whereupon it recycles. This mode generates a square wave. See mode 3 on 8254 data sheet.
SOFTWARESTROBE :	Output of counter (OUT N) pulses low for one clock cycle on terminal count. Count starts after counter is loaded. See mode 4 on 8254 data sheet.
HARDWARESTROBE :	Output of counter (OUT N) pulses low for one clock cycle on terminal count. Count starts on rising edge at GATE N input. See mode 5 on 8254 data sheet.

Returns: Error code or 0 if no errors

cbC8536Config()

Description: Configures 8536 counter for desired operation. This function can only be used with 8536 counters.

Summary: int cbC8536Config (int BoardNum, int CounterNum, int OutputControl, int RecycleMode, int Retrigger)

Arguments: BoardNum - May be 0 to 99 (0 to 9 for 16 bit version of Universal Library).
CounterNum - counter number to configure
OutputControl - Specifies counter output signal used.
RecycleMode - Execute once or reload and re-execute until stopped.
Retrigger - Enable or disable retriggering.

EXPLANATION OF THE ARGUMENTS:
BoardNum - refers to the number associated with the board when it was installed with the configuration program.

CounterNum - Selects one of the counter channels. An 8536 has 3 counters. The value may be 1, 2 or 3.

OutputControl - Specifies the action of the output signal. The options for this argument are:
HIGHPULSEONTC - Output will transition from low to high for one clock pulse on terminal count.
TOGGLEONTC - Output will change state on terminal count.
HIGHUNTILTC - Output will transition to high at the start of counting then go low on terminal count.

RecycleMode - If set to RECYCLE (as opposed to ONETIME) then the counter will automatically reload to the starting count every time it reaches 0, then counting will continue.

Retrigger - If set to CBENABLED then every trigger on the counter's trigger input will initiate loading of the initial count and counting will proceed from initial count.

Returns: Error code or 0 if no errors

cbC8536Init()

Description: Initializes the counter linking features of an 8536 counter chip. See the 8536 data sheet Counter/Timer Link Controls section for a complete description of the hardware affected by this mode. The linking of counters 1 & 2 must be accomplished prior to enabling the counters.

Summary: int cbC8536Init (int BoardNum, int ChipNum, int CtrlOutput)

Arguments: BoardNum - May be 0 to 99 (0 to 9 for 16 bit version of Universal Library).
ChipNum - chip number to configure
CtrlOutputl - Specifies counter output signal used.

EXPLANATION OF THE ARGUMENTS:

BoardNum - Board number of board with 8536 counter installed.

ChipNum - Selects one of the 8536 chips on the board, 1 to n.

CtrlOutput - Specifies how the counter 1 is to be linked to counter 2, if at all. The options for this argument are:
NOTLINKED -Counter 1 is not connected to any other counters inputs.
GATECTR2 - Output of counter 1 is connected to the GATE of counter #2.
TRIGCTR2 - Output of counter 1 is connected to the trigger of counter #2.
INCTR2 - Output of counter 1 is connected to counter #2 clock input.

Returns - Error code or 0 if no errors.

cbC9513Config()

Description:	Sets all of the configurable options of a 9513 counter.
Summary:	int cbC9513Config (int BoardNum, int CounterNum, int GateControl, int CounterEdge, int CountSource, int SpecialGate, int Reload, int RecycleMode, int BCDMode, int CountDirection, int OutputControl)
Arguments:	BoardNum - May be 0 to 99 (0 to 9 for 16 bit version of Universal Library). CounterNum - counter number (1 - n) GateControl - gate control CountEdge - which edge to count CountSource - which of the available count sources to use. SpecialGate - special gate may be enabled or disabled. Reload - Load or load and hold. RecycleMode - Execute once or reload and recycle. BCDMode - Counter may operate in Binary Coded Decimal if desired. CountDirection - AM9513 may count up or down. OutputControl - The type of output desired.

EXPLANATION OF THE ARGUMENTS:

ADVICE

The information provided here and in cb9513Init() will only help you understand how Universal Library syntax corresponds to the 9513 data sheet. It is not a substitute for the data sheet. You cannot program and use a 9513 without the manufacturers' data book.

BoardNum - refers to the board number associated with the board when it was installed with the configuration program. The specified board must have a 9513 counter.

CounterNum - counter number (1 - n) where n is the number of counters on the board. (For example, a CIO-CTR5 has 5, a CIO-CTR10 has 10, etc. See board-specific information.)

GateControl - gate control variables are:

Universal Library Syntax	Corresponds to 9513 description in Counter Mode Register Description
NOGATE	No gating
AHLTCPREVCTR	Active high TCN-1
AHLNEXTGATE	Active High Level GATE N + 1
AHLPREVGATE	Active High Level GATE N - 1
AHLGATE	Active High Level GATE N
ALLGATE	Active Low Level GATE N
AHEGATE	Active High Edge GATE N
ALEGATE	Active Low Edge GATE N

CountEdge - which edge to count. Refer to Source Edge in 9513 data book.

Universal Library Syntax	Corresponds to 9513 description in Counter Mode Register Description
POSITIVEEDGE	Count on Rising Edge
NEGATIVEEDGE	Count on Falling Edge

CountSource -

Universal Library Syntax	Corresponds to 9513 description in Counter Mode Register Description
TCPREVCTR	TCN - 1 (Terminal count of previous counter)
CTRINPUT1	SRC 1 (Counter Input 1)
CTRINPUT2	SRC 2 (Counter Input 2)
CTRINPUT3	SRC 3 (Counter Input 3)
CTRINPUT4	SRC 4 (Counter Input 4)
CTRINPUT5	SRC 5 (Counter Input 5)
GATE1	GATE 1
GATE2	GATE 2
GATE3	GATE 3
GATE4	GATE 4
GATE5	GATE 5
FREQ1	F1
FREQ2	F2
FREQ3	F3
FREQ4	F4
FREQ5	F5

SpecialGate -

Universal Library Syntax	Corresponds to 9513 description in Counter Mode Register Description
CBENABLED	Enable Special Gate
CBDISABLED	Disable Special Gate

Reload -

Universal Library Syntax	Corresponds to 9513 description in Counter Mode Register Description
LOADREG	Reload from Load
LOADANDHOLDREG	Reload from Load or Hold except in Mode X which reloads only from Load

RecycleMode -

Universal Library Syntax	Corresponds to 9513 description in Counter Mode Register Description
ONETIME	Count Once
RECYCLE	Count Repetitively

BCDMode -

Universal Library Syntax	Corresponds to 9513 description in Counter Mode Register Description
CBDISABLED	Binary Count
CBENABLED	BCD Count

CountDirection -

Universal Library Syntax	Corresponds to 9513 description in Counter Mode Register Description
COUNTDOWN	Count Down
COUNTUP	Count Up

OutputControl -

Universal Library Syntax	Corresponds to 9513 description in Counter Mode Register Description

ALWAYSLOW	Inactive, Output Low
HIGHPULSEONTC	High pulse on Terminal Count
TOGGLEONTC	TC Toggled
DISCONNECTED	Inactive, Output High Impedance
LOWPULSEONTC	Active Low Terminal Count Pulse
3, 6, 7 (numeric values)	Illegal

Returns: error code or 0 if no errors

cbC9513Init()

Description: Initializes all of the chip level features of a 9513 counter chip. This function can only be used with 9513 counters.

Summary: int cbC9513Init (int BoardNum, int ChipNum, int FoutDivider, int FoutSource, int Compare1, int Compare2, int TimeOfDay)

Arguments: BoardNum - May be 0 to 99 (0 to 9 for 16 bit version of Universal Library).
ChipNum - Specifies which 9513 chip is to be initialized.
FoutDivider - F-Out divider (0-15)
FoutSource - Specifies source of the signal for F-Out signal.
Compare1 - CBENABLED or CBDISABLED
Compare2 - CBENABLED or CBDISABLED
TimeOfDay - CBDISABLED or 1-3

EXPLANATION OF THE ARGUMENTS:

BoardNum - refers to the board number associated with the board when it was installed with the configuration program. The specified board must have a 9513 counter.

ChipNum - Specifies which 9513 chip is to be initialized. For a CTR05 board this should be set to 1. For a CTR10 board it should be either 1 or 2, and for a CTR20 it should be 1-4.

FoutDivider -

Universal Library Syntax	Corresponds to 9513 description in Counter Mode Register Description
0	Divide by 16
1	Divide by 1
2 ... 15	Divide by the number 2 ... 15

FoutSource -

Universal Library Syntax	Corresponds to 9513 description in Counter Mode Register Description	
CTRINPUT1	SRC 1	(Counter Input 1)
CTRINPUT2	SRC 2	(Counter Input 2)
CTRINPUT3	SRC 3	(Counter Input 3)
CTRINPUT4	SRC 4	(Counter Input 4)
CTRINPUT5	SRC 5	(Counter Input 5)
GATE1	GATE 1	
GATE2	GATE 2	
GATE3	GATE 3	
GATE4	GATE 4	
GATE5	GATE 5	
FREQ1	F1	
FREQ2	F2	
FREQ3	F3	
FREQ4	F4	
FREQ5	F5	

Compare1 -

Universal Library Syntax	Corresponds to 9513 description in Counter Mode Register Description
CBDISABLED	Disabled
CBENABLED	Enabled

Compare2 -

Universal Library Syntax	Corresponds to 9513 description in Counter Mode Register Description
CBDISABLED	Disabled
CBENABLED	Enabled

TimeOf Day -

Universal Library Syntax	Corresponds to 9513 description in Counter Mode Register Description
CBDISABLED	TOD Disabled
1	TOD Enabled / 5 Input
2	TOD Enabled / 6 Input
3	TOD Enabled / 10 Input

No Arguments - For:

Universal Library Set To	Corresponds to 9513 description in Counter Mode Register Description
0 (FOUT on)	FOUT Gate
0 (Data bus matches board)	Data Bus Width
1 (Disable Increment)	Data Pointer Control
1 (BCD Scaling)	Scalar Control

Returns: Error code or 0 if no errors

cbCFreqIn()

Description: Measures the frequency of a signal. This function can only be used with 9513 counters. This function uses internal counters #5 and #4.

Summary: int cbCFreqIn (int BoardNum, int SigSource, int GateInterval, unsigned *Count, long *Freq)

Arguments: BoardNum - May be 0 to 99 (0 to 9 for 16 bit version of Universal Library).
SigSource - specifies which signal will be measured
GateInterval - gating interval in milliseconds (must be >0)
Count - The raw count is returned here
Freq - the measured frequency in Hz returned here.

EXPLANATION OF THE ARGUMENTS:

BoardNum - refers to the board number associated with the board when it was installed with the configuration program. The specified board must have a 9513 counter.

SigSource - specifies the source of the signal from which the frequency will be calculated.

The signal to be measured is routed internally from the source specified by SigSource to the clock input of counter 5. On boards with more than one 9513 chip, there is more than one counter 5. Which counter 5 is used is also determined by SigSource. SigSource may be set to one of the following values:

One 9513 chip:	
CTRINPUT1 through CTRINPUT5	Chip 1 used
GATE1 through GATE4	
FREQ1 through FREQ5	
Two 9513 chips:	
CTRINPUT1 through CTRINPUT10	Chip 1 or Chip 2 used
GATE 1 through GATE 9 (excluding gate 5)	
FREQ1 through FREQ10	
Four 9513 chips:	
CTRINPUT1 through CTRINPUT20	Chips 1- 4 may be used
GATE1 through GATE19 (excluding gates 5, 10 & 15)	
FREQ1 through FREQ20	

The value of SigSource determines which chip will be used. CTRINPUT6 through CTRINPUT10, FREQ6 through FREQ10 and GATE6 through GATE9 indicate chip two will be used. The signal to be measured must be present at the chip two input specified by SigSource. Also, the gating connection from counter 4 output to counter 5 gate must be made between counters 4 and 5 OF THIS CHIP (see below).

See board specific information to determine valid values for your board.

GateInterval - specifies the time (in milliseconds) that the counter will be counting. The optimum GateInterval depends on the frequency of the measured signal. The counter can count up to 65535.

If the gating interval is too low, then the count will be too low and the resolution of the frequency measurement will be poor. For example, if the count changes from 1 to 2 the measured frequency doubles.

If the gating interval is too long then the counter will overflow and a FREQOVERFLOW error will occur.

This function will not return until the GateInterval has expired. There is no background option. Under Windows this means that window activity will stop for the duration of the call. Adjust the GateInterval so this does not pose a problem to your user interface.

Returns: Error code or 0 if no errors
Count - Count that frequency calculation based on returned here
Freq - Measured frequency in Hz returned here

Note: This function requires an electrical connection between counter 4 output and counter 5 gate. This connection must be made between counters 4 and 5 ON THE CHIP DETERMINED BY SIGSOURCE.

Also, cb9513Init() must be called for each ChipNum that will be used by this function. The values of FoutDivider, FoutSource, Compare1, Compare2, and Timeof Day are irrelevant to this function and may be any value shown in the cbC9513Init function description.

cbCIn()

Description: Reads the current count from a counter

Summary: int cbCIn (int BoardNum, int CounterNum, unsigned int *Count)

Arguments: BoardNum - May be 0 to 99 (0 to 9 for 16 bit version of Universal Library).
CounterNum - counter number to read
Count - Count returned here

EXPLANATION OF THE ARGUMENTS:

BoardNum - refers to the board number associated with the board when it was installed with the configuration program. The specified board must have a counter.

CounterNum - The counter to read current count from. Valid values are 1 to 20, up to the number of counters on the board.

1	Read from counter 1
2 .. 20	Read from counter n (n = 1...20)

Returns: Error code or 0 if no errors
Count - counter value returned here . The range of counter values returned are:
0 to 65,535 for C or PASCAL languages.
Please see notes on BASIC integer data types in your BASIC manual.
-32,768 to 32,767 for BASIC languages. BASIC reads counters as:

-1	reads as	65,535
-32,768	reads as	32,768
32,767	reads as	32,767
2	reads as	2
0	reads as	0

cbCIn32()

Description: Reads the current count from a counter and returns it as a 32 bit integer.

Summary: int cbCIn32 (int BoardNum, int CounterNum, unsigned long *Count)

Arguments: BoardNum - May be 0 to 99 (0 to 9 for 16 bit version of Universal Library).
CounterNum - Counter number(1 - n) to read
Count - Current count is returned here

EXPLANATION OF THE ARGUMENTS:

BoardNum - Refers to the board number associated with the board when it was installed with the configuration program. The specified board must have an LS7266 counter.

CounterNum - The counter to read current count from. Valid values are 1 to N, where N is the number of counters on the board.

Count - Current count value from selected counter is returned here

Returns: Error code or 0 if no error occurs

Note: cbCIn() vs cbCIn32()
The cbCIn() and cbCIn32() perform the same operation. The only difference between the two is that cbCIn() returns a 16 bit count value and cbCIn32() returns a 32 bit value. The only time you need to use cbCIn32() is when reading counters that are larger than 16 bits. The only boards that have such counters are the quadrature encoder input boards (CIO-QUAD02, CIO-QUAD04, PCM-QUAD02). For these boards both cbCIn() and cbCIn32() can be used but cbCIn32 is required whenever you need to read count values greater than 16 bits (counts > 65535).

cbCLoad()

Description: Loads the specified counter's LOAD, HOLD, ALARM, COUNT, PRESET or PRESCALER register with a count. When you want to load a counter with a value to count from, it is never loaded directly into the counter's count register. It is loaded into the load or hold register. From there, the counter, once enabled, loads the count from the appropriate register, generally on the first valid pulse.

Summary: int cbCLoad (int BoardNum, int RegName, unsigned int LoadValue)

Arguments: BoardNum - May be 0 to 99 (0 to 9 for 16 bit version of Universal Library).
RegName - Register to load LoadValue into.
LoadValue - Value to be loaded into RegName

EXPLANATION OF THE ARGUMENTS:

BoardNum - refers to the board number associated with the board when it was installed with the configuration program. The specified board must have a counter.

RegName - The register to load the count to. Valid values are:

LOADREG1 .. 20	Load registers 1 through 20. This may span several chips.
HOLDREG1 .. 20	Hold registers 1 through 20. This may span several chips. (9513 only)
ALARM1CHIP1	Alarm register 1 of the first counter chip. (9513 only)
ALARM2CHIP1	Alarm register 2 of the first counter chip. (9513 only)
ALARM1CHIP2	Alarm register 1 of the second counter chip. (9513 only)
ALARM2CHIP2	Alarm register 2 of the second counter chip. (9513 only)
ALARM1CHIP3	Alarm register 1 of the third counter chip. (9513 only)
ALARM2CHIP3	Alarm register 2 of the third counter chip. (9513 only)
ALARM1CHIP4	Alarm register 1 of the four counter chip. (9513 only)
ALARM2CHIP4	Alarm register 2 of the four counter chip. (9513 only)
COUNT1 .. 4	Current Count (LS7266 only)
PRESET1 .. 4	Preset register (LS7266 only)
PRESCALER1 .. 4	Prescaler register (LS7266 only)

LoadValue - The value to be loaded. Must be between 0 and 2 ^ resolution - 1 of the counter. For example, a 16 bit counter is 2 ^ 16 - 1, or 65,535 (Please see notes on Basic integer types...).

Note: You cannot load a count-down-only counter with less than 2.

Returns: Error code or 0 if no errors

Counter Types: There are several counter types supported. Please refer to the data sheet for the registers available for a counter type.

cbCLoad32()

Description: Loads the specified counter's COUNT, PRESET or PRESCALER register with a count.

Summary: int cbCLoad32 (int BoardNum, int RegName, unsigned long LoadValue)

Arguments: BoardNum - May be 0 to 99 (0 to 9 for 16 bit version of Universal Library).
RegName - Register to load LoadValue in to.
LoadValue - Value to be loaded into RegName

EXPLANATION OF THE ARGUMENTS:

BoardNum - Refers to the board number associated with the board when it was installed with the configuration program. The specified board must have an LS7266 counter.

RegName - The register to load the value into. Valid register names are:

COUNT1 - 4	Used to initialize the counter
PRESET1 - 4	Used to set upper limit of counter in some modes
PRESCALER1 - 4	Used for clock filtering

LoadValue - The value to be loaded.

Returns: Error code or 0 if no error occurs

Note: cbCLoad() vs cbCLoad32()
The cbCLoad() and cbCLoad32() perform the same operation. The only difference between the two is that cbCLoad() loads a 16 bit count value and cbCLoad32() loads a 32 bit value. The only time you need to use cbCLoad32() is when loading counts that are larger than 32 bits (counts > 65535).

cbCStoreOnInt() Changed R4.0 RW

Description: Installs an interrupt handler that will store the current count whenever an interrupt occurs. This function can only be used with 9513 counters. This function will continue to operate in the background until either IntCount has been satisfied or cbStopBackground is called.

Summary: DOS Language:
int cbCStoreOnInt (int BoardNum, int IntCount, int CntrControl[], unsigned CountData[])
Windows Language:
int cbCStoreOnInt (int BoardNum, int IntCount, int CntrControl[], int MemHandle[])

Arguments: BoardNum - May be 0 to 99 (0 to 9 for 16 bit version of Universal Library).
IntCount - Number of interrupts
CntrControl - Array with each element set to either CBENABLED or CBDISABLED
DataBuffer - Array where counts will be stored (DOS)
MemHandle - Handle for Windows buffer. (Windows)
In VEE there is no Count Data on this panel. The counter data is in the cbvGetCtrStatus panel. See the section HP VEE Specific Functions for an explanation.

EXPLANATION OF THE ARGUMENTS:

BoardNum - refers to the board number associated with the board when it was installed with the configuration program. The specified board must have a 9513 counter.

IntCount - The counters will be read every time an interrupt occurs until IntCount interrupts have occurred. If IntCount is = 0 then the function will run until cbStopBackground is called. (See CountData details).

CntrControl - The array should have an element for each counter on the board. (5 elements for CTR-05 board, 10 elements for a CTR-10, etc.). Each element corresponds to a possible counter channel. Each element should be set to either CBDISABLED or CBENABLED. All channels that are set to CBENABLED will be read when an interrupt occurs.

CountData - The array should have an element for each counter on the board. (5 elements for CIO-CTR05 board, 10 elements for a CIO-CTR10, etc.). Each element corresponds to a possible counter channel. Each channel that is marked as CBENABLED in the CntrControl array will be read when an interrupt occurs. The count value will be stored in the DataBuffer element associated with that channel.

> **New Functionality**: If the Library Revision is set to 4.0 or greater then the following code changes are required.
>
> If IntCount is non-zero then the CountData array must be allocated to (IntCount * Number of Counters).
>
> For example, if IntCount is set to 100 for a CIO-CTR05 board, then the CountData array must be declared with a size of (100 * 5) = 500. This new functionality keeps the user application from having to move the data out of the CountData buffer for every interrupt, before it is overwritten. Now, for each interrupt the counter values will be stored in adjacent memory locations in the CountData array.

Note: *Specifying IntCount to be a non-zero value and failing to allocate the proper sized array will result in a runtime error. There is no way for the Universal Library to determine if the array has been allocated with the proper size.*

If IntCount = 0 the functionality is unchanged.

Returns: Error code or 0 if no errors

cbCStatus()

Description: Returns status information about the specified counter (7266 counters only)

Summary: int cbCStatus (int BoardNum, int CounterNum, unsigned long *StatusBits)

Arguments: BoardNum - May be 0 to 99 (0 to 9 for 16 bit version of Universal Library).
CounterNum - Counter number(1 - n) to read
StatusBits - Status information is returned here

EXPLANATION OF THE ARGUMENTS:

BoardNum - Refers to the board number associated with the board when it was installed with the configuration program. The specified board must have an LS7266 counter.

CounterNum - The counter to read current count from. Valid values are 1 to N, where N is the number of counters on the board.

StatusBits - Current status from selected counter is returned here. The status consists of individual bits that indicate various conditions within the counter. The currently defined status bits are:

C_UNDERFLOW	- Is set to 1 whenever the count decrements past 0. Is cleared to 0 whenever cbCGetStatus() is called.
C_OVERFLOW	- Is set to 1 whenever the count increments past it's upper limit. Is cleared to 0 whenever cbCGetStatus() is called.
C_COMPARE	- Is set to 1 whenever the count matches the preset register. Is cleared to 0 whenever cbCGetStatus() is called.
C_SIGN	- Is set to 1 when the MSB of the count is 1. Is cleared to 0 whenever the MSB of the count is set to 0.
C_ERROR	- Is set to 1 whenever an error occurs due to excessive noise on the input. Is cleared to 0 by calling cbC7266Config().
C_UP_DOWN	- Is set to 1 when counting up. Is cleared to 0 when counting down
C_INDEX	- Is set to 1 when index is valid. Is cleared to 0 when index is not valid.

Returns: Error code or 0 if no error occurs**.**

cbDBitIn()

Description: Reads the state of a single digital input bit. This function treats all of the DIO ports on a board as a single very large port. It lets you read the state of any individual bit within this large port. If the port type is not AUXPORT you must use cbDConfig() to configure the port for input first.

Summary: int cbDBitIn (int BoardNum, int PortType, int BitNum, int *BitValue);

Arguments: BoardNum - May be 0 to 99 (0 to 9 for 16 bit version of Universal Library).
PortType - Specifies which type of digital port to read
BitNum - Specifies which bit to read
BitValue - Place holder for return value of bit - the bit's value (0 or 1) is returned here

EXPLANATION OF THE ARGUMENTS:
BoardNum - refers to the number associated with the board when it was installed with the configuration program.

PortType- There are two general types of digital I/O - 8255 and other. Some boards (DIO Series) use an 8255 for digital I/O. For these boards PortType should be set to FIRSTPORTA. Other boards don't use 8255. For these boards PortType should be set to AUXPORT. Some boards have both types of digital I/O (DAS1600). Set Port-Num to either FIRSTPORTA or AUXPORT depending on which digital inputs you wish to read.

BitNum - This specifies the bit number within the single large port. The specified bit must be in a port that is currently configured as an input.

The tables below show which bit numbers are in which 82C55 and 8536 digital chips. The most 82C55 chips on a single board is eight (8), on the CIO-DIO196. The most (2) 8536 chips occur on the CIO-INT32.

82C55 Bit#	Chip #	Address	8536 Bit#	Chip #	Address
0 - 23	1	Base + 0	0 - 19	1	Base + 0
24 - 47	2	Base + 4	20 - 39	2	Base + 4
48 - 71	3	Base + 8			
72 - 96	4	Base + 12			
96 - 119	5	Base + 16			
120 - 143	6	Base + 20			
144 - 167	7	Base + 24			
168 - 191	8	Base + 28			

BitValue - Place holder for return value of bit. Value will be 0 or 1. A 0 indicates a low reading, a 1 indicates a logic high reading. Logic high does not necessarily mean 5V. See the board manual for chip input specifications.

Returns: Error code or 0 if no errors
BitValue - value (0 or 1) of specified bit returned here

cbDBitOut()

Description: Sets the state of a single digital output bit. This function treats all of the DIO chips on a board as a single very large port. It lets you set the state of any individual bit within this large port. If the port type is not AUXPORT, you must use cbDConfig() to configure the port for output first.

Summary: int cbDBitOut (int BoardNum, int PortType, int BitNum, int BitValue);

Arguments: BoardNum - May be 0 to 99 (0 to 9 for 16 bit version of Universal Library).
PortNum - Specifies which digital port (AUXPORT, FIRSTPORTA).
BitNum - specifies which bit to write
BitValue - the bit's value (0 or 1)

EXPLANATION OF THE ARGUMENTS:
BoardNum - refers to the number associated with the board when it was installed with the configuration program.

PortType - There are two general types of digital I/O - 8255 and other. Some boards (DIO Series) use an 8255 for digital I/O. For these boards PortType should be set to FIRSTPORTA. Other boards don't use 8255. For these boards PortType should be set to AUXPORT. Some boards have both types of digital I/O (CIO-DAS1600). Set PortNum to either FIRSTPORTA or AUXPORT depending on which digital inputs you wish to write.

BitNum - This specifies the bit number within the single large port. The specified bit must be in a port that is currently configured as an output.

The tables below show which bit numbers are in which 82C55 and 8536 digital chips. The most 82C55 chips on a single board is eight (8), on the CIO-DIO196. The most (2) 8536 chips occur on the CIO-INT32.

82C55 Bit#	Chip #	Address
0 - 23	1	Base + 0
24 - 47	2	Base + 4
48 - 71	3	Base + 8
72 - 96	4	Base + 12
96 - 119	5	Base + 16
120 - 143	6	Base + 20
144 - 167	7	Base + 24
168 - 191	8	Base + 28

8536 Bit#	Chip #	Address
0 - 19	1	Base + 0
20 - 39	2	Base + 4

BitValue - The value to set the bit to. Value will be 0 or 1. A 0 indicates a logic low output, a 1 indicates a logic high output. Logic high does not necessarily mean 5V. See the board manual for chip specifications.

Returns: Error code or 0 if no errors

cbInByte() / cbInWord()

Description: Reads a byte or word from a hardware register on a board.

Summary: int cbInByte (int BoardNum, int PortNum)
int cbInWord (int BoardNum, int PortNum)

Arguments: BoardNum - May be 0 to 99 (0 to 9 for 16 bit version of Universal Library).
PortNum - register on the board to read

EXPLANATION OF THE ARGUMENTS:
BoardNum - refers to the board number associated with the board when it was installed with the configuration program.

PortNum - register within the board. Boards are set to a particular base address. The registers on the boards are at addresses that are offsets from the base address of the board (BaseAdr+0, BaseAdr+2, etc). This argument should be set to the offset for the desired register. This function takes care of adding the base address to the offset, so that the board's address can be changed without changing the code.

NOTES:
cbInByte() is used to read 8 bit ports. cbInWord() is used to read 16 bit ports.

Returns: The current value of the specified register

cbOutByte() / cbOutWord()

Description: Writes a byte or word to a hardware register on a board.

Summary: int cbOutByte (int BoardNum, int PortNum, int PortVal)
int cbOutWord (int BoardNum, int PortNum, int PortVal)

Arguments: BoardNum - May be 0 to 99 (0 to 9 for 16 bit version of Universal Library).
PortNum - register on the board to write to
PortVal - value to write to register

EXPLANATION OF THE ARGUMENTS:
BoardNum - refers to the board number associated with the board when it was installed with InstaCal.

PortNum - register within the board. Boards are set to a particular base address. The registers on the boards are at addresses that are offsets from the base address of the board (BaseAdr+0, BaseAdr+2, etc). This argument should be set to the offset for the desired register. This function takes care of adding the base address to the offset, so that the board's address can be changed without changing the code.

PortVal - Value that will be written to the register

NOTE: cbOutByte() is used to write to 8 bit ports. cbOutWord() is used to write to 16 bit ports.

Returns: error code or 0 if no errors

cbDConfigPort()

Description: Configures a digital port as Input or Output. This mode is for use with 82C55 chips and 8536 chips. See the board user's manual for details of chip operation.

Summary:	int cbDConfigPort (int BoardNum, int PortNum, int Direction)
Arguments:	BoardNum - May be 0 to 99 (0 to 9 for 16 bit version of Universal Library).
	PortNum - Specifies which digital I/O port to configure.
	Direction - DIGITALOUT or DIGITALIN

EXPLANATION OF THE ARGUMENTS:

PortNum - The specified port must be configurable. The AUXPORT is not configurable. It is always configured for inputs and outputs.

The tables below show which ports and bit numbers are in which 82C55 and 8536 digital chips. The most 82C55 chips on a single board is eight (8), on the CIO-DIO196. The most (2) 8536 chips occur on the CIO-INT32.

Mnemonic	Bit#	8255 Port #	Port Address	8536 Port #	Port Address
FIRSTPORTA	0 - 7	1A	Base + 0	1A	Base + 0
FIRSTPORTB	8 - 15	1B		1B	
FIRSTPORTCL	16 - 19	1CL		1C	
FIRSTPORTCH	20 - 23	1CH		Not present	
SECONDPORTA	24 - 31	2A	Base + 4	2A	Base + 4
SECONDPORTB	32 - 39	2B		2B	
SECONDPORTCL	40 - 43	2CL		2C	
SECONDPORTCH	44 - 47	2CH		No port C High in 8536 chips	

and so on to the last chip on the board as: THIRDPORT@, FOURTHPORT@, FIFTHPORT@, SIXTHPORT@, SEVENTHPORT@ and

EIGHTHPORTA	168 - 175	8A	Base + 28
EIGHTHPORTB	176 - 183	8B	
EIGHTHPORTCL	184 - 187	8CL	
EIGHTHPORTCH	188 - 191	8CH	

Direction - DIGITALOUT or DIGITALIN configures entire eight or four bit port for output or input.

Returns: Error code or 0 if no errors

Note: Using this function will reset all ports on a chip configured for output to a zero state. This means that if you set an output value on FIRSTPORTA and then change the configuration on FIRSTPORTB from OUTPUT to INPUT, the output value at FIRSTPORTA will be all zeros. You can, however, set the configuration on SECOND-PORTX without affecting the value at FIRSTPORTA. For this reason, this function is usually called at the beginning of the program for each port requiring configuration.

cbDIn()

Description: Reads a digital input port

Summary: int cbDIn (int BoardNum, int PortNum, int *DataValue);

Arguments: BoardNum - May be 0 to 99 (0 to 9 for 16 bit version of Universal Library).
PortNum - Specifies which digital I/O port to read.
*DataValue - Digital input value returned here.

EXPLANATION OF THE ARGUMENTS:
BoardNum - refers to the number associated with the board when it was installed with the configuration program.

PortNum - If the port type is not AUXPORT, the specified port must be configured for input. The AUXPORT is not configurable.

The tables below show which ports are in which 82C55 and 8536 digital chips. The most 82C55 chips on a single board is eight (8), on the CIO-DIO196. The most (2) 8536 chips occur on the CIO-INT32.

Mnemonic	8255 Port #	Chip Address	8536 Port #	Chip Address
FIRSTPORTA	1A	Base + 0	1A	Base + 0
FIRSTPORTB	1B		1B	
FIRSTPORTCL	1CL		1C	
FIRSTPORTCH	1CH		Not present	
SECONDPORTA	2A	Base + 4	2A	Base + 4
SECONDPORTB	2B		2B	
SECONDPORTCL	2CL		2C	
SECONDPORTCH	2CH		No port C High in 8536 chips	

and so on to the last chip on the board as: THIRDPORT@, FOURTHPORT@, FIFTHPORT@, SIXTHPORT@, SEVENTHPORT@ and

EIGHTHPORTA	8A	Base + 28
EIGHTHPORTB	8B	
EIGHTHPORTCL	8CL	
EIGHTHPORTCH	8CH	

The size of the ports vary. If it is an eight bit port then the returned value will be in the range 0- 255. If it is a four bit port the value will be in the range 0 -15.

Returns: Error code or 0 if no errors
*DataValue - Digital input value returned here

IMPORTANT NOTE: Be sure to look at the example programs and the board specific information contained in the Universal Library User's Guide for clarification of valid PortNum values.

cbDInScan()

Description: Multiple reads of digital input port of a high speed digital port on a board with a pacer clock such as the CIO-PDMA16.

Summary: DOS Language:
int cbDInScan (int BoardNum, int PortNum, long Count, long *Rate, int DIData[], int Options)
Windows Language:
int cbDInScan (int BoardNum, int PortNum, long Count, long *Rate, int MemHandle, int Options)

Arguments: BoardNum - May be 0 to 99 (0 to 9 for 16 bit version of Universal Library).
PortNum - Specifies which digital I/O port to read
Count - number of times to read digital input
*Rate - Number of times per second (Hz) to read
DIData - Digital input values returned here. (DOS)
MemHandle - Handle for Windows buffer. (Windows)
In VEE this panel is called Data Array. See VEE specific information for more details.
Options - Bit fields that control various options

EXPLANATION OF THE ARGUMENTS:
BoardNum - refers to the number associated with the board when it was installed with the configuration program.

PortNum - Specifies which digital I/O port to read (usually, FIRSTPORTA or FIRSTPORTB). The specified port must be configured as an input.

Count - The number of times to read digital input.

***Rate** - Number of times per second (Hz) to read the port. The actual sampling rate in some cases will vary a small amount from the requested rate. The actual rate will be returned to the Rate argument.

DIData - The data array must be at least large enough to hold Count integers and must be capable of holding 16 bit values if the Option WORDXFER is specified.

MemHandle - Handle for Windows buffer to store data in (Windows). This buffer must have been previously allocated with the cbWinBufAlloc() function.

Options:
BACKGROUND - If the BACKGROUND option is not used then the cbDInScan() function will not return to your program until all of the requested data has been collected and returned to DataBuffer.

When the BACKGROUND option is used, control will return immediately to the next line in your program and the transfer from the digital input port to DataBuffer will continue in the background. Use cbGetStatus() to check on the status of the background operation. Use cbStopBackground() to terminate the background process before it has completed.

CONTINUOUS - This option puts the function in an endless loop. Once it transfers the required number of bytes it resets to the start of DataBuffer and begins again. The only way to stop this operation is with cbStopBackground(). Normally this option should be used in combination with BACKGROUND so that your program will regain control.

EXTCLOCK - If this option is used then transfers will be controlled by the signal on the trigger input line rather than by the internal pacer clock.   Each transfer will be triggered on the appropriate edge of the trigger input signal (see board-specific information).   When this option is used the Rate argument is ignored. The transfer rate is dependent on the trigger signal.

WORDXFER - Normally this function reads a single (byte) port. If WORDXFER is specified then it will read two adjacent ports on each read and store the value of both ports together as the low and high byte of a single array element in DataBuffer[].

Transfer Method - May not be specified. DMA is used.

Returns: Error code or 0 if no errors.
*Rate - Actual sampling rate returned here.
DataBuffer[] - Digital input value returned here.

cbDOut()

Description: Writes a byte to a digital output port .

Summary: int cbDOut (int BoardNum, int PortNum, int DataValue);

Arguments: BoardNum - May be 0 to 99 (0 to 9 for 16 bit version of Universal Library).
PortNum - Specifies which digital I/O port write.
*DataValue - Digital input value to be written.

EXPLANATION OF THE ARGUMENTS:
PortNum - If the port type is not AUXPORT, the specified port must be configured for output. The AUXPORT is not configurable.

BoardNum - refers to the number associated with the board when it was installed with the configuration program.

The tables below show which ports are in which 82C55 and 8536 digital chips. The most 82C55 chips on a single board is eight (8), on the CIO-DIO196. The most (2) 8536 chips occur on the CIO-INT32.

Mnemonic	8255 Port #	Chip Address	8536 Port #	Chip Address
FIRSTPORTA	1A	Base + 0	1A	Base + 0
FIRSTPORTB	1B		1B	
FIRSTPORTCL	1CL		1C	
FIRSTPORTCH	1CH		Not present	
SECONDPORTA	2A	Base + 4	2A	Base + 4
SECONDPORTB	2B		2B	
SECONDPORTCL	2CL		2C	
SECONDPORTCH	2CH		No port C High in 8536 chips and so on to	

the last chip on the board as: THIRDPORT@, FOURTHPORT@, FIFTHPORT@, SIXTHPORT@, SEVENTHPORT@ and

EIGHTHPORTA	8A	Base + 28
EIGHTHPORTB	8B	
EIGHTHPORTCL	8CL	
EIGHTHPORTCH	8CH	

The size of the ports vary. If it is an eight bit port then the output value should be in the range 0- 255. If it is a four bit port the value should be in the range 0 - 15.

Returns: Error code or 0 if no errors.

IMPORTANT NOTE: Be sure to look at the example programs and the board specific information in the Universal Library User's Guide for clarification of valid PortNum values.

cbDOutScan()

Description: Multiple writes to digital output port of a high speed digital port on a board with a pacer clock such as the CIO-PDMA16.

Summary: DOS Language:
int cbDOutScan (int BoardNum, int PortNum, long Count, long *Rate, int DOData[], int Options)
Windows Language:
int cbDOutScan (int BoardNum, int PortNum, long Count, long *Rate, int MemHandle, int Options)

Arguments: BoardNum - May be 0 to 99 (0 to 9 for 16 bit version of Universal Library).
PortNum - Specifies which digital I/O port to write
Count - number of times to write digital output
*Rate - Number of times per second (Hz) to write
DOData - Digital output values. (DOS)
MemHandle - Handle for Windows buffer. (Windows)
In VEE this panel is called Data Array. See VEE specific information for more details
Options - Bit fields that control various options

Defaults: BYTEXFER option is the default. Make sure you are using an array where your data is arranged in bytes, not words. There is a WORDXFER option for word array transfers.

EXPLANATION OF THE ARGUMENTS:

PortNum - Specifies which digital I/O port to write. The two choices are FIRSTPORTA or FIRSTPORTB. The specified port must be configured as an output.

Count - The number of times to write digital output.

***Rate** - Number of times per second (Hz) to write to the port. The actual update rate in some cases will vary a small amount from the requested rate. The actual rate will be returned to the Rate argument.

DOData - The data array must be at least large enough to hold Count integers and must be capable of holding 16 bit values if the Option WORDXFER is specified.

MemHandle - Handle for Windows buffer to store data in (Windows). This buffer must have been previously allocated with the cbWinBufAlloc() function.

Options:
BACKGROUND - If the BACKGROUND option is not used then the cbDOutScan() function will not return to your program until all of the requested data has been output.

When the BACKGROUND option is used, control will return immediately to the next line in your program and the transfer to the digital output port from DataBuffer will continue in the background. Use cbGetStatus() to check on the status of the background operation. Use cbStopBackground() to terminate the background process before it has completed.

CONTINUOUS - This option puts the function in an endless loop. Once it transfers the required number of bytes it resets to the start of DOData and begins again. The only way to stop this operation is with cbStopBackground(). Normally this option should be used in combination with BACKGROUND so that your program will regain control.

EXTCLOCK - If this option is used then transfers will be controlled by the signal on the trigger input line rather than by the internal pacer clock. Each transfer will be triggered on the appropriate edge of the trigger input signal (see board specific information). When this option is used the Rate argument is ignored. The transfer rate is dependent on the trigger signal.

WORDXFER - Normally this function writes a single (byte) port. If WORDXFER is specified then it will write two adjacent ports as the low and high byte of a single array element in DOData[].

Transfer Method - May not be specified. DMA is used.

Returns:	Error code or 0 if no errors.
	*Rate - Actual sampling rate returned here.
	DataBuffer[] - Array containing the data to be written.

cbErrHandling()

Description: Sets the error handling for all subsequent function calls. Most functions return error codes after each call. In addition other error handling features have been built into the library. This function controls those features. If the Universal Library cannot find the configuration file CB.CFG, it always terminates the program regardless of the cbErrHandling() setting.

Summary: int cbErrHandling (int ErrReporting, int ErrHandling)

Arguments: ErrReporting - type of error reporting.
ErrHandling - type of error handling.

EXPLANATION OF THE ARGUMENTS:

Warnings vs Fatal Errors - All errors that can occur are classified as either "warnings" or "fatal". Errors that can occur in normal operation in a bug free program (disk is full, too few samples before trigger occurred) are classified as "warnings". All other errors indicate a more serious problem and are classified as "fatal".

ErrReporting - This argument controls when the library will print error messages on the screen. The default is DONTPRINT. If it is set to:

DONTPRINT - Errors will not generate a message to the screen. In that case your program must always check the returned error code after each library call to determine if an error occurred.

PRINTWARNINGS - Only warning errors will generate a message to the screen. Your program will have to check for fatal errors.

PRINTFATAL - Only fatal errors will generate a message to the screen. Your program must check for warning errors.

PRINTALL - All errors will generate a message to the screen.

ErrHandling - This argument specifies what class of error will cause the program to halt. The default is DONTSTOP. If ErrReporting is set to:

DONTSTOP - The program will always continue executing when an error occurs.

STOPFATAL - The program will halt if a "fatal" error occurs.

STOPALL - Will stop whenever any error occurs. If you are running in an Integrated Development Environment (IDE) then when errors occur, the environment may be shut down along with the program. If your IDE behaves this way (QuickBasic and VisualBasic do) then you should set ErrHandling to DONTSTOP. You can check error codes to determine the cause of the error.

Returns: - Always returns to 0.

HP VEE

The ErrHandling argument is ignored by HP VEE. The parameter is automatically set to DONT STOP. To stop a running VEE program use a conditional object to detect and decode error messages from Universal Library and call a VEE Stop object to terminate the program if an error is detected that should terminate the program. This is not how we desire the function to work and are trying to improve the communication between Universal Library and VEE so that full error handling is enabled.

cbFileAInScan()

Description: Scan a range of A/D channels and store the samples in a disk file. This function reads the specified number of A/D samples at the specified sampling rate from the specified range of A/D channels from the specified board. If the A/D board has programmable gain then it sets the gain to the specified range. The collected data is returned to a file in binary format. Use cbFileRead() to load data from that file into an array. See board-specific information to determine if this function is supported on your board.

Summary: int cbFileAInScan (int BoardNum, int LowChan, int HighChan, long Count, long *Rate, int Range, char *FileName, unsigned Options)

Arguments: BoardNum - May be 0 to 99 (0 to 9 for 16 bit version of Universal Library).
LowChan - First A/D channel of scan
HighChan - Last A/D channel of scan
Count - Number of samples to collect
Rate - Sample rate in samples per second (Hz) per channel
Range - Range code
FileName - Name of disk file
Options - Bit fields that control various options

EXPLANATION OF THE ARGUMENTS:

BoardNum - refers to the board number associated with the board when it was installed with the configuration program. The specified board must have an A/D.

Low / High Chan - Indicates the range of channels to scan. LowChan must be less than or equal to HighChan. The maximum allowable channel depends on which type of A/D board is being used. For boards that have both single ended and differential inputs the maximum allowable channel number also depends on how the board is configured. For example, a CIO-DAS1600 has 8 chans for differential, 16 for single ended.

Count - Specifies the total number of A/D samples that will be collected. If more than one channel is being sampled then the number of samples collected per channel is equal to Count / (HighChan - LowChan+1).

Rate - The maximum sampling rate depends on the A/D board that is being used (see Rate description cbAInScan).

Range - If the selected A/D board does not have a programmable range feature, then this argument will be ignored. Otherwise the gain can be set to any of the following ranges that are supported by the selected A/D board. Refer to board specific information for the list of ranges supported by each board.

BIP10VOLTS	+/- 10 volts	UNI10VOLTS	0 to 10 volts
BIP5VOLTS	+/- 5 volts	UNI5VOLTS	0 to 5 volts
BIP2PT5VOLTS	+/- 2.5 volts	UNI2PT5VOLTS	0 to 2.5 volts
BIP1PT67VOLTS	+/- 1.67 volts	UNI1PT67VOLTS	0 to 1.67 volts
BIP1PT25VOLTS	+/- 1.25 volts	UNI2VOLTS	0 to 2 volts
BIP1VOLTS	+/- 1 volts	UNI1PT25VOLTS	0 to 1.25 volts
BIPPT625VOLTS	+/- 0.625 volts	UNI1VOLTS	0 to 1 volts
BIPPT5VOLTS	+/- 0.5 volts	UNIPT1VOLTS	0 to 0.1 volts
BIPPT1VOLTS	+/- 0.1 volts	UNIPT01VOLTS	0 to 0.01 volts

BIPPT05VOLTS	+/-0.05 volts	MA4TO20	4 to 20 mA
BIPPT01VOLTS	+/- 001 volts	MA2TO10	2 to 10 mA
BIPPT005VOLTS	+/- 0.005 volts	MA1TO5	1 to 5 mA
		MAPT5TO2PT5	0.5 to 2.5 mA

FileName - The name of the file in which data will be stored. When using the 16 bit version of the Universal Library, the named file must already exist. It should have been previously created with the MAKESTRM.EXE program.

Options:
EXTCLOCK - If this option is used then conversions will be controlled by the signal on the trigger input line rather than by the internal pacer clock. Each conversion will be triggered on the appropriate edge of the trigger input signal (see board-specific information). When this option is used the Rate argument is ignored. The sampling rate is dependent on the trigger signal.

EXTTRIGGER - If this option is specified the sampling will not begin until the trigger condition is met. If this option is specified the sampling will not begin until the trigger condition is met. On many boards, this trigger condition is programmable (see cbSetTrigger() function and board-specific information for details). On other boards, only 'polled gate' triggering is supported. In this case assuming active high operation, data acquisition will commence immediately if the trigger input is high. If the trigger input is low, acquisition will be held off until it goes high. Acquisition will then continue until NumPoints& samples have been taken regardless of the state of the trigger input. This option is most useful if the signal is a pulse with a very low duty cycle (trigger signal in TTL low state most of the time) so that triggering will be held off until the occurrence of the pulse.

DTCONNECT - Samples are sent to the DT-Connect port if the board is equipped with one.

Returns: Error code or 0 if no errors
*Rate = actual sampling rate

OVERRUN Error - This error indicates that the data was not written to the file as fast as the data was sampled. Consequently some data was lost. The value returned from cbFileGetInfo in *TotalCount will be the number of points that were successfully collected.

VERY IMPORTANT NOTE

In order to understand the functions, you must read the Board Specific Information section found in the Universal Library user's guide. The example programs should be examined and run prior to attempting any programming of your own. Following this advice will save you hours of frustration, and possibly time wasted holding for technical support.

This note, which appears elsewhere, is especially applicable to this function. Now is the time to read the board specific information for your board. We suggest that you make a copy of that page to refer to as you read this manual and examine the example programs.

cbFileGetInfo()

Description: Returns information about a streamer file. When cbFileAInScan() or cbFilePretrig() fills the streamer file, information is stored about how the data was collected (sample rate, channels sampled etc.). This function returns that information. See board-specific information to determine if this function is supported on your board.

Summary: int cbFileGetInfo (char *FileName, int *LowChan, int *HighChan, long *PreTrigCount, int *TotalCount, long *Rate, int *Range)

Arguments:	FileName - Name of streamer file
	LowChan - Variable to return LowChan to
	HighChan - Variable to return HighChan to
	PreTrigCount - Variable to return PreTrigCount to
	TotalCount - Variable to return TotalCount to
	Rate - Variable to return sampling rate to
	Range - Variable to return A/D range code to
Returns:	Error code or 0 if no errors
	*LowChan - low A/D channel of scan
	*HighChan - high A/D channel of scan
	*TotalCount - total number of points collected
	*PreTrigCount - number of pre-trigger points collected
	*Rate - sampling rate when data was collected
	*Range - Range of A/D when data was collected

cbFilePretrig()

Description: Scan a range of channels continuously while waiting for a trigger. Once the trigger occurs, return the specified number of samples including the specified number of pre-trigger samples to a disk file. This function waits for a trigger signal to occur on the Trigger Input. Once the trigger occurs, it returns the specified number (TotalCount) of A/D samples including the specified number of pre-trigger points. It collects the data at the specified sampling rate (Rate) from the specified range (LowChan-HighChan) of A/D channels from the specified board. If the A/D board has programmable gain then it sets the gain to the specified range. The collected data is returned to a file. See board-specific information to determine if this function is supported by your board.

Summary: int cbFilePretrig (int BoardNum, int LowChan, int HighChan, long *PreTrigCount, long *TotalCount, long *Rate, int Range, char *FileName, unsigned Options)

Arguments: BoardNum - May be 0 to 99 (0 to 9 for 16 bit version of Universal Library).
LowChan - First A/D channel of scan
HighChan - Last A/D channel of scan
PreTrigCount - Number of pre-trigger samples to collect
TotalCount - Total number of samples to collect
Rate - Sample rate in samples per second (Hz) per channel
Range - A/D Range
FileName - Name of disk file
Options - Bit fields that control various options

EXPLANATION OF THE ARGUMENTS:

BoardNum - refers to the board number associated with the board when it was installed with the configuration program. The specified board must have an A/D and pretrigger capability.

Channel - The maximum allowable channel depends on which type of A/D board is being used. For boards that have both single ended and differential inputs the maximum allowable channel number also depends on how the board is configured (8 chans for differential, 16 for single ended).

PreTrigCount - Specifies the number of samples before the trigger that will be returned. PreTrigCount must be less than 16000 and PreTrigCount must also be less than TotalCount-512.

If the trigger occurs too early, then fewer than the requested number of pre-trigger samples will be collected. In that case a TOOFEW error will occur. The PretrigCount will be set to indicate how many samples were collected and the post trigger samples will still be collected.

TotalCount - Specifies the total number of samples that will be collected and stored in the file. TotalCount must be greater than or equal to PretrigCount+512. If the trigger occurs too early then fewer than the requested number of samples will be collected. In that case a TOOFEW error will occur. The TotalCount will be set to indicate how many samples were actually collected.

Rate - The maximum sampling rate depends on the A/D board that is being used. This is the rate at which scans are triggered. If you are sampling 4 channels, 0-3, then specifying a rate of 10,000 scans per second (10KHz) will result in the A/D converter rate of 40KHz: 4 channels at 10,000 samples per channel per second. This is different from some software where you specify the total A/D chip rate. In those systems, the per channel rate is equal to the A/D rate divided by the number of channels in a scan. This argument also returns the value of the actual set. This may be different from the requested rate because of pacer limitations.

Range - If the selected A/D board does not have a programmable range feature, then this argument is ignored. Otherwise the gain can be set to any of the following ranges that are supported by the selected A/D board. Refer to board specific information for the list of ranges supported by each board.

BIP10VOLTS	+/- 10 volts	UNI10VOLTS	0 to 10 volts
BIP5VOLTS	+/- 5 volts	UNI5VOLTS	0 to 5 volts
BIP2PT5VOLTS	+/- 2.5 volts	UNI2PT5VOLTS	0 to 2.5 volts
BIP1PT67VOLTS	+/- 1.67 volts	UNI1PT67VOLTS	0 to 1.67 volts
BIP1PT25VOLTS	+/- 1.25 volts	UNI2VOLTS	0 to 2 volts
BIP1VOLTS	+/- 1 volts	UNI1PT25VOLTS	0 to 1.25 volts
BIPPT625VOLTS	+/- 0.625 volts	UNI1VOLTS	0 to 1 volts
BIPPT5VOLTS	+/- 0.5 volts	UNIPT1VOLTS	0 to 0.1 volts
BIPPT1VOLTS	+/- 0.1 volts	UNIPT01VOLTS	0 to 0.01 volts
BIPPT05VOLTS	+/-0.05 volts	MA4TO20	4 to 20 mA
BIPPT01VOLTS	+/- 001 volts	MA2TO10	2 to 10 mA
BIPPT005VOLTS	+/- 0.005 volts	MA1TO5	1 to 5 mA
		MAPT5TO2PT5	0.5 to 2.5 mA

FileName - The named file must already exist. It should have been previously created with the MAKESTRM.EXE program.

Options:
EXTCLOCK - If this option is used then conversions will be controlled by the signal on the trigger input line rather than by the internal pacer clock. Each conversion will be triggered on the appropriate edge of the trigger input signal (see board-specific information). When this option is used the Rate argument is ignored. The sampling rate is dependent on the trigger signal.

DTCONNECT - Samples are sent to the DT-Connect port if the board is equipped with one.

Returns: error code or 0 if no errors
*PreTrigCount - actual number of pre-trigger samples collected
*TotalCount - actual number of samples collected
*Rate = actual sampling rate

OVERRUN Error - This error indicates that the data was not written to the file as fast as the data was sampled. Consequently some data was lost. The value in TotalCount will be the number of points that were successfully collected.

cbFileRead()

Description: Reads data from a streamer file. See board-specific information to determine if this function is supported on your board.

Summary int cbFileRead (char *FileName, long FirstPoint, long *TotalCount, int *DataBuffer)

Arguments: FileName - Name of streamer file
FirstPoint - Index of first point to read
TotalCount - Number of points to read from file
DataBuffer - Pointer to data buffer that data will be read into.

Data Format - The data is returned as 16 bits. The 16 bits may represent 12 bits of analog, 12 bits of analog plus 4 bits of channel, or 16 bits of analog. Use cbConvertData() to correctly load the data into an array.

Loading Portions of Files - The file may contain much more data than can fit in DataBuffer. In those cases use TotalCount and FirstPoint to read a selected piece of the file into DataBuffer. Call cbFileGetInfo() first to find out how many points are in the file.

Returns: Error code or 0 if no errors
DataBuffer - data read from file
Total Count - number of points actually read. This may be less than the requested number of points if an error occurs.

cbFromEngUnits()

Description:	Converts a voltage (or current) in engineering units to a D/A count value for output to a D/A.
Summary:	int cbFromEngUnits (int BoardNum, int Range, float EngUnits, unsigned *DataVal)
Arguments:	BoardNum - May be 0 to 99 (0 to 9 for 16 bit version of Universal Library). Range - D/A range to use in conversion EngUnits - Voltage (or current) value to convert DataVal - D/A count equivalent to voltage returned here

EXPLANATION OF THE ARGUMENTS

BoardNum - refers to the board number associated with the D/A board when it was installed. This function uses the board number to determine the range and resolution values to use in the conversion.

Range - D/A voltage (or current) range. Some D/A boards have programmable voltage ranges, others set the voltage range via switches on the board. In either case the selected range must be passed to this function. Each D/A board supports different voltage and/or current ranges. Refer to the board's hardware manual for a list of allowed ranges used by the board.

EngUnits - The voltage (or current) value that you wish to set the D/A to. This value should be within the range specified by the Range argument.

DataVal - The function returns a D/A count to this variable that is equivalent to the EngUnits argument.

cbToEngUnits()

Description: Converts an A/D count value to an equivalent voltage value.

Summary: int cbToEngUnits (int BoardNum, int Range, unsigned DataVal, float *EngUnits)

Arguments: BoardNum - May be 0 to 99 (0 to 9 for 16 bit version of Universal Library).
Range - A/D range to use in conversion
DataVal - A/D count value returned from an A/D board
EngUnits - Equivalent voltage (or current) value returned to this variable

EXPLANATION OF THE ARGUMENTS

BoardNum - refers to the board number associated with the A/D board when it was installed. This function uses the board number to determine the range and resolution values to use for the conversion.

Range - A/D voltage (or current) range. Some A/D boards have programmable voltage ranges, others set the voltage range via switches on the board. In either case the selected range must be passed to this function. Each A/D board supports different voltage and/or current ranges. Refer to the board's hardware manual for a list of allowed ranges used by the board.

DataVal - A/D count returned from an A/D board.

EngUnits - The voltage (or current) value that is equivalent to DataVal is returned to this variable. The value will be within the range specified by the Range argument.

cbGetBoardName()

Description: Returns the board name of a specified board.

Summary: int cbGetBoardName (int Board, char *BoardName);

Arguments: Board -May be 0 to 99 (0 to 9 for 16 bit version of Universal Library), or GETFIRST or GETNEXT
BoardName - Board name string returned to this variable

EXPLANATION OF THE ARGUMENTS

Board - refers either to the board number associated with a board when it was installed OR GETFIRST or GETNEXT.

BoardName - A string variable that the board name will be returned to. This string variable must be pre-allocated to be at least as large as BOARDNAMELEN. This size is guaranteed to be large enough to hold the longest board name string.

NOTES:
There are two distinct ways of using this function. The first is to pass a board number as the Board argument. In that case the string that is returned will describe the board type of the installed board. When using this method to iterate through a list of boards that may or may not be installed, call cbGetConfig(BOARDINFO, BoardNum, 0, BIBOARDTYPE, ConfigVal) first with BoardNum incremented on each call, calling cbGetBoardName() only when the value returned in ConfigVal is not 0.

The other way to use the function is to set Board to GETFIRST or GETNEXT to get a list of all board types that are supported by the library. If Board is set to GETFIRST it will return the first board type in the list of supported boards. Subsequent calls with Board=GETNEXT will return each of the other board types supported by the library. When you reach the end of the list BoardName will be set to an empty string. Refer to the ulgt04 example programs for more details.

cbGetConfig()

Description: Returns a configuration option for a board. The configuration information for all boards is stored in the CB.CFG file. This information is loaded from CB.CFG by all programs that use the library. The current configuration can be changed within a running program with the cbSetConfig() function. The cbGetConfig() function returns the current configuration information.

Summary: int cbGetConfig (int InfoType, int BoardNum, int DevNum, int ConfigItem, int *ConfigVal)

Arguments: InfoType - Which class of configuration information you want to retrieve
BoardNum - May be 0 to 99 (0 to 9 for 16 bit version of Universal Library).
DevNum - Specifies which device within board
ConfigItem - Specifies which configuration item
ConfigVal - Current configuration value returned here

EXPLANATION OF THE ARGUMENTS

InfoType - The configuration information for each board is grouped into different categories. This argument specifies which category you want. It should be set to one of the following constants:
GLOBALINFO - Information about the configuration file
BOARDINFO - general information about a board
DIGITALINFO - information about a digital device
COUNTERINFO - information about a counter device
EXPANSIONINFO - information about an expansion device
MISCINFO - One of the miscellaneous options for the board

BoardNum - refers to the board number associated with a board when it was installed.

DevNum - Selects a particular device. If InfoType=DIGITALINFO then DevNum specifies which of the board's digital devices you want information on. If InfoType=COUNTERINFO then DevNum specifies which of the board's counter devices you want information on.

ConfigItem - Specifies which configuration item you wish to retrieve. Refer to the table below for a list of all of the possible values for ConfigItem.

ConfigVal - The specified configuration item is returned to this variable.

NOTES:
The list of ConfigItem values for each category of configuration information is:

InfoType = GLOBALINFO
GIVERSION - CB.CFG file format. This information is used by the library to determine compatibility.
GINUMBOARDS - Maximum number of installable boards
GINUMEXPBOARDS - Maximum number of expansions boards allowed to be installed.

InfoType = BOARDINFO
BIBASEADR - Base address of board

BIBOARDTYPE - Returns a unique number in the range of 0 to 8000 Hex describing the board type installed.

BIINTLEVEL - Interrupt level. 0 for none or 1 - 15

BIDMACHAN - DMA channel. 0, 1 or 3

BIINITIALIZED - TRUE (non-zero) or FALSE (0)

BICLOCK - Clock frequency in MHz (1, 4, 6 or 10) or 0 for not supported.

BIRANGE - Selected voltage range. For switch-selectable gains only.

BIRANGE: If the selected A/D board does not have a programmable gain feature then this argument returns the range as defined by the installed InstaCal settings, which, if InstaCal and the board were installed correctly, corresponds to the input range as set via the switches on the board. Refer to board specific information for a list of the A/D ranges supported by each board.

Library Name	Range	BIRANGE #	Library Name	Range	BIRANGE #
BIP10VOLTS	+/- 10 volts	1	UNI10VOLTS	0 to 10 volts	100
BIP5VOLTS	+/- 5 volts	0	UNI5VOLTS	0 to 5 volts	101
BIP2PT5VOLTS	+/- 2.5 volts	2	UNI2PT5VOLTS	0 to 2.5 volts	102
BIP1PT67VOLTS	+/- 1.67 volts	3	UNI1PT67VOLTS	0 to 1.67V	103
BIP1VOLTS	+/- 1 volts	4	UNI1PT25VOLTS	0 to 1.25V	104
BIPPT625VOLTS	+/- 0.625 volts	5	UNI1VOLTS	0 to 1 volts	105
BIPPT5VOLTS	+/- 0.5 volts	6	UNIPT1VOLTS	0 to 0.1 volts	106
BIPPT1VOLTS	+/- 0.1 volts	7	UNIPT01VOLTS	0 to 0.01V	107
BIPPT05VOLTS	+/-0.05 volts	8	UNIPT67VOLTS	0 to 1.67V	108
BIPPT01VOLTS	+/- 001 volts	9	MA4TO20	4 to 20 mA	200
BIPPT005VOLTS	+/- 0.005 volts	10	MA2TO10	2 to 10 mA	201
BIP1PT67VOLTS	+/-1.67 volts	11	MA1TO5	1 to 5 mA	202
			MAPT5TO2PT5	0.5 to 2.5mA	203

BINUMADCHANS - Number of A/D channels

BIUSESEXPS - Supports expansion boards TRUE/FALSE

BIDINUMDEVS - Number of digital devices

BIDIDEVNUM - Index into digital information for first device

BICINUMDEVS - Number of counter devices

BICIDEVNUM - Index into counter information for first device

BINUMDACHANS - Number of D/A channels

BIWAITSTATE - Setting of Wait State jumper. 1 = enabled, 0 = disabled

BINUMIOPORTS - Number of I/O ports used by board

BIPARENTBOARD - Board number of parent board

BIDTBOARD - Board number of connected DT board

InfoType = DIGITALINFO

DIBASEADR - Base address

DIINITIALIZED - TRUE (non-zero) or FALSE (0)

DIDEVTYPE - Device Type - AUXPORT, FIRSTPORTA etc

DIMASK - Bit mask for this port

DIREADWRITE - Read required before write TRUE/FALSE

DICONFIG - Current configuration INPUT or OUTPUT

DINUMBITS - Number of bits in port

DICURVAL - Current value of outputs

InfoType = COUNTERINFO

CIBASEADR - Base address
CIINITIALIZED - TRUE (non-zero) or FALSE (0)
CICTRTYPE - 1 = 8254, 2 = 9513 , 3 = 8536, 4 = 7266 type counter chip.
CICTRNUM - Which counter on chip
CICONFIGBYTE - Configuration byte

InfoType = EXPANSIONINFO

XIBOARDTYPE - Board type
XIMUXADCHAN1 - A/D channel board is connect to
XIMUXADCHAN2 - 2nd A/D channel board is connected to
XIRANGE1 - Range (gain) of low 16 chans
XIRANGE2 - Range (gain) of high 16 chans
XICJCCHAN - A/D channel that CJC is connected to
XITHERMTYPE - Thermocouple type
XINUMEXPCHANS - Number of expansion channels on board
XIPARENTBOARD - Board number of parent A/D board

cbGetErrMsg()

Description: Returns the error message associated with an error code. Each function returns an error code. If the error code is not equal to 0 it indicates that an error occurred. Call this function to convert the returned error code to a descriptive error message.

Summary: int cbGetErrMsg (int ErrCode, char ErrMsg[ERRSTRLEN])

Arguments: ErrCode - error code that was returned by any function in library .
ErrMsg - error message returned here

The ErrMsg variable must be pre-allocated to be at least as large as ERRSTRLEN. This size is guaranteed to be large enough to hold the longest error message.

See also cbErrHandling() for an alternate method of handling errors.

Returns: Error code or 0 if no errors
*ErrMsg - error message string is returned here

cbGetStatus()

Description:	Returns status about background operation currently running.
Summary:	int cbGetStatus (int BoardNum, int *Status, long *CurCount, long *CurIndex)
Arguments:	BoardNum - May be 0 to 99 (0 to 9 for 16 bit version of Universal Library). Status - pointer to where status will be returned CurCount - current count returned to this variable. CurIndex - current index returned to this variable.

EXPLANATION OF THE ARGUMENTS:

Status - status indicates whether or not a background process is currently executing.

CurCount - Specifies how many points have been input or output. It can be used to gauge how far along the operation is towards completion. Generally the CurCount will return the total number of samples collected at the time of the call to cbGetStatus(). However, in cases where CONTINUOUS and BACKGROUND options are both set, the way that CurCount behaves will depend on board type and transfer mode. This value may recycle as the circular buffer recycles, or may continuously increment with the number of counts transferred. Also, CurCount may not update on each sample. For example, when running in BLOCKIO mode, CurCount updates after each packet of data has been transferred. The packet size is board dependent. Refer to board specific information for details.

CurIndex - This is an index into the data buffer that points at the start of the last completed channel scan. This can be used to provide a real- time display for a background operation. DataBuffer[CurIndex] points to the start of the last complete channel scan that was put in or taken out of the buffer. You should expect CurIndex to increment by the number of channels in the scan as well. If no points in the buffer have been accessed yet then CurIndex will equal -1. This value can also behave differently in cases where CONTINUOUS and BACKGROUND options are both set (see CurCount description). Refer to board specific information for details.

If you use the CONVERTDATA option with either the CONTINUOUS option or with pre-triggering functions then CurIndex will return the index of the last A/D sample, rather than the start of the last completed channel scan.

For many background operations CurCount = CurIndex. For Pre-Trigger inputs though, they are different. If the hardware allows background trigger operations, CurCount indicates how many points of the TotalCount have been collected. CurCount will rise to PreTrigCount, stop until the trigger occurs then rise to TotalCount. CurIndex though will constantly increase and reset as it goes around and around the circular buffer while waiting for the trigger to occur.

Returns:	Error code or 0 if no errors *Status= IDLE - No background operation has been executed RUNNING - Background operation still underway *CurCount = current number of samples collected *CurIndex = Current sample index

HP VEE

VEE Programs Stopping Background Tasks Early
The red STOP button on the cbGetStatus panel must be used when stopping background processes before the scheduled completion. If instead you use the stop button on the VEE icon bar, the background process will continue to run in the background. The result of doing so and exiting VEE is undefined. Please do not fail to use the cbGetStatus() STOP button. See example programs ULAI03.VEE through ULAI06.VEE.

cbMemRead()

Description: Reads data from a memory board into an array.

Summary: int cbMemRead (int BoardNum, unsigned DataBuffer[], long FirstPoint, long Count)

Arguments: BoardNum - May be 0 to 99 (0 to 9 for 16 bit version of Universal Library).
DataBuffer - Pointer to the data array
FirstPoint - Index of first point to read or FROMHERE.
Count - Number of points (words) to read

EXPLANATION OF THE ARGUMENTS:

BoardNum - refers to the board number associated with the board when it was installed with the configuration program.

FirstPoint - Use the FirstPoint argument to specify the first point to be read. For example, to read points #200 - #250, set FirstPoint=200 and Count=50.

If you are going to read a large amount of data from the board in small chunks then set FirstPoint to FROMHERE to read each successive chunk. Using FROMHERE speeds up the operation of cbMemRead() when working with large amounts of data.

For example, to read 300,000 points in 100,000 point chunks the calls would look like this

```
cbMemRead (0, DataBuffer, 0, 100000)
cbMemRead (0, DataBuffer, FROMHERE, 1000000)
cbMemRead (0, DataBuffer, FROMHERE, 1000000)
```

DT-CONNECT Conflicts - The cbMemRead() function can not be called while a DT-CONNECT transfer is in progress. For example, if you start collecting A/D data to the memory board in the background (by calling cbAInScan() with the DTCONNECT + BACKGROUND options) you can not call cbMemRead() until the cbAInScan has completed. If you do you will get a DTACTIVE error.

Returns: Error code or 0 if no errors
DataBuffer - data read from memory board

cbMemReadPretrig()

Description: Reads pre-trigger data from a memory board that has been collected with the cbAPretrigger() function and re-arranges the data in the correct order (pre-trigger data first, then post-trigger data). This function can only be used to retrieve data that has been collected with the cbAPretrigger() function with EXTMEMORY set in the options argument. After each cbAPretrigger() call, all data must be unloaded from the memory board with this function. If any more data is sent to the memory board then the pre-trigger data will be lost.

Summary: int cbMemReadPretrig (int BoardNum, unsigned DataBuffer[], long FirstPoint, long Count)

Arguments:	BoardNum - May be 0 to 99 (0 to 9 for 16 bit version of Universal Library).
	DataBuffer - Pointer to the data array
	FirstPoint - Index of first point to read or FROMHERE.
	Count - Number of points (words) to read

EXPLANATION OF THE ARGUMENTS:

BoardNum - refers to the board number associated with the board when it was installed with the configuration program

FirstPoint - Use the FirstPoint argument to specify the first point to be read. For example, to read points #200 - #250, set FirstPoint=200 and Count=50.

If you are going to read a large amount of data from the board in small chunks then set FirstPoint to FROMHERE to read each successive chunk. Using FROMHERE speeds up the operation of cbMemReadPretrig() when working with large amounts of data.

For example, to read 300,000 points in 100,000 chunks the calls would look like this

```
        cbMemReadPretrig (0, DataBuffer, 0, 100000)
        cbMemReadPretrig (0, DataBuffer, FROMHERE, 1000000)
        cbMemReadPretrig (0, DataBuffer, FROMHERE, 1000000)
```

DT Connect Conflicts - The cbMemReadPretrig() function can not be called while a DT Connect transfer is in progress. For example, if you start collecting A/D data to the memory board in the background (by calling cbAInScan() with the DTCONNECT + BACKGROUND options), you can not call cbMemReadPretrig() until the cbAInScan has completed. If you do you will get a DTACTIVE error.

Returns:	Error code or 0 if no errors
	DataBuffer - data read from memory board

cbMemReset()

Description: Resets the memory board pointer to the start of the data. The memory boards are sequential devices. They contain a counter which points to the 'current' word in memory. Every time a word is read or written this counter increments to the next word.

Summary: int cbMemReset (int BoardNum)

Arguments: BoardNum - May be 0 to 99 (0 to 9 for 16 bit version of Universal Library).

This function is used to reset the counter back to the start of the memory. Between successive calls to cbAInScan() you would call this function so that the second cbAInScan() overwrites the data from the first call. Otherwise the data from the first cbAInScan() will be followed by the data from the second cbAInScan() in the memory on the card.

Likewise, anytime you call cbMemRead() or cbMemWrite() it will leave the counter pointing to the next memory location after the data that you read or wrote. Call cbMemReset() to reset back to the start of the memory buffer before the next call to cbAInScan().

Returns: Error code or 0 if no errors

cbMemSetDTMode()

Description: Sets the DT Connect Mode of a Memory Board

Summary: int cbMemSetDTMode (int BoardNum, int Mode)

Arguments: BoardNum - May be 0 to 99 (0 to 9 for 16 bit version of Universal Library).
Mode - Direction of memory board DT Transfer

EXPLANATION OF THE ARGUMENTS:

Mode - Must be set to either DTIN or DTOUT. Set the Mode on the memory board to DTIN if you wish to transfer data from an A/D board to the memory board. Set Mode=DTOUT if you wish to transfer data from a memory board to a D/A board.

This command only controls the direction of data transfer between the memory board and another board that is connected to it via a DT Connect cable.

If using the EXTMEMORY option for cbAInScan(), etc., this function should not be used. The Memory Board mode is already set through the EXTMEMORY option. Use this function only if the parent board is not supported by the Universal Library.

cbMemWrite()

Description: Writes data from an array to the memory card

Summary: int cbMemWrite(int BoardNum, unsigned DataBuffer[], long FirstPoint, long Count)

Arguments: BoardNum - May be 0 to 99 (0 to 9 for 16 bit version of Universal Library).
DataBuffer - Pointer to the data array
FirstPoint - Index of first point to write or FROMHERE.
Count - Number of points (words) to write

EXPLANATION OF THE ARGUMENTS:

FirstPoint - Use the FirstPoint argument to specify where in the board's memory to write the first point. For example, to write to locations #200 - #250, set FirstPoint=200 and Count=50.

If you are going to write a large amount of data to the board in small chunks then set FirstPoint to FROMHERE to write each successive chunk. Using FROMHERE speeds up the operation of cbMemWrite() when working with large amounts of data.

For example, to write 300,000 points in 100,000 point chunks the calls would look like this

```
cbMemWrite (0, DataBuffer, 0, 100000)
cbMemWrite (0, DataBuffer, FROMHERE, 100000)
cbMemWrite (0, DataBuffer, FROMHERE, 100000)
```

DT Connect Conflicts - The cbMemWrite() function can not be called while a DT Connect transfer is in progress. For example, if you start collecting A/D data to the memory board in the background (by calling cbAInScan() with the DTCONNECT + BACKGROUND options). You can not call cbMemWrite() until the cbAInScan has completed. If you do you will get a DTACTIVE error.

Returns: Error code or 0 if no errors

cbRS485()

Description: Sets direction of RS485 communications port buffers.

Summary: int cbRS485 (int BoardNum, int Transmit, int Receive)

Arguments: BoardNum - May be 0 to 99 (0 to 9 for 16 bit version of Universal Library).
Transmit - set to CBENABLED or CBDISABLED
Receive - set to CBENABLED or CBDISABLED

EXPLANATION OF THE ARGUMENTS:

Transmit: The transmit RS485 line driver is turned on. Data written to the RS485 UART chip will be transmitted to the cable connected to that port.

Receive: The receive RS485 buffer is turned on. Data present on the cable connected to the RS485 port will be received by the UART chip.

Both the transmit and receive buffers may be enabled or disabled simultaneously. If both are enabled, data written to the port will also be received by the port. For a complete discussion of RS485 network construction and communication, please refer to the CIO-COM485 or PCM-COM485 hardware manual.

Returns: None

cbSetConfig

Description: Sets a configuration option for a board. The configuration information for all boards is stored in the CB.CFG file. All programs that use the library read this file. This function can be used to override the configuration information stored in the CB.CFG file.

Summary: int cbSetConfig (int InfoType, int BoardNum, int DevNum, int ConfigItem, int ConfigVal);

Arguments: InfoType - Which class of configuration information you want to set
BoardNum - May be 0 to 99 (0 to 9 for 16 bit version of Universal Library).
DevNum - Specifies which device within board
ConfigItem - Specifies which configuration item
ConfigVal - New value to set option to

EXPLANATION OF THE ARGUMENTS

InfoType - The configuration information for each board is grouped into different categories. This argument specifies which category you want. It should be set to one of the following constants:

BOARDINFO - general information about a board
DIGITALINFO - information about a digital device
COUNTERINFO - information about a counter device
EXPANSIONINFO - information about an expansion device
MISCINFO - One of the miscellaneous options for the board

BoardNum - refers to the board number associated with a board when it was installed.

DevNum - Selects a particular device. If InfoType=DIGITALINFO then DevNum specifies which of the board's digital devices you want to set information on. If InfoType=COUNTERINFO then DevNum specifies which of the board's counter devices.

ConfigItem - Specifies which configuration item you wish to set. Refer to the table below for a list of all of the possible values for ConfigItem.

ConfigVal - The value to set the specified configuration item to.

NOTES:
The list of ConfigItem values for each category of configuration information is:

InfoType = BOARDINFO

BIBASEADR - Base address of board
BIBOARDTYPE - Board Type
BIINTLEVEL - Interrupt level
BIDMACHAN - DMA channel
BIINITIALIZED - TRUE (non-zero) or FALSE (0)
BICLOCK - Clock frequency in MHz (1, 4, 6 or 10)
BIRANGE - Selected voltage range
BINUMADCHANS - Number of A/D channels
BIUSESEXPS - Supports expansion boards TRUE/FALSE

BIDINUMDEVS - Number of digital devices
BIDIDEVNUM - Index into digital information for first device
BICINUMDEVS - Number of counter devices
BICIDEVNUM - Index into counter information for first device
BINUMDACHANS - Number of D/A channels
BIWAITSTATE - Setting of Wait State jumper
BINUMIOPORTS - Number of I/O ports used by board
BIPARENTBOARD - Board number of parent board
BIDTBOARD - Board number of connected DT board

InfoType = DIGITALINFO
DIBASEADR - Base address
DIINITIALIZED - TRUE (non-zero) or FALSE (0)
DIDEVTYPE - Device Type - AUXPORT, FIRSTPORTA etc
DIMASK - Bit mask for this port
DIREADWRITE - Read require before write TRUE/FALSE
DICONFIG - Current configuration INPUT or OUTPUT
DINUMBITS - Number of bits in port
DICURVAL - Current value of outputs

InfoType = COUNTERINFO
CIBASEADR - Base address
CIINITIALIZED - TRUE (non-zero) or FALSE (0)
CICTRTYPE - 8254, 8536, 7266 or 9513 counter
CICTRNUM - Which counter on chip
CICONFIGBYTE - Configuration byte

InfoType = EXPANSIONINFO
XIBOARDTYPE - Board type
XIMUXADCHAN1 - A/D channel board is connect to
XIMUXADCHAN2 - 2nd A/D channel board is connected to
XIRANGE1 - Range (gain) of low 16 chans
XIRANGE2 - Range (gain) of high 16 chans
XICJCCHAN - A/D channel that CJC is connected to
XITHERMTYPE - Thermocouple type
XINUMEXPCHANS - Number of expansion channels on board
XIPARENTBOARD - Board number of parent A/D board

cbSetTrigger()

Description: This function is used to select the trigger source and setup its parameters. This trigger is used to initiate analog to digital conversions using the following Universal Library functions:

- cbAInScan, if the EXTRIGGER option is selected.
- cbAPretrig
- cbFilePretrig

Summary: int cbSetTrigger(int BoardNum, int Type, unsigned LowThreshold, unsigned HighThreshold)

Arguments: BoardNum - May be 0 to 99 (0 to 9 for 16 bit version of Universal Library).
Type - trigger type (see table below)
LowThreshold - low threshold for analog trigger
HighThreshold - high threshold for analog trigger

EXPLANATION OF THE ARGUMENTS

int *BoardNum*

Specifies the board number associated with the board when it was installed with the configuration program. The board must have the software selectable triggering source and/or options.

int *Type*

Specifies the type of triggering based on the external trigger source. This can be one of the constants specified in the column labeled in the column labeled Type in this table.

TRIGGER SOURCE	TYPE	EXPLANATION
Analog	GATE_NEG_HYS	AD conversions are enabled when the external analog trigger input is more positive than HighThreshold. AD conversions are disabled when the external analog trigger input more negative than Low/Threshold. Hysterisis is the level between Low/Threshold and HighThreshold.
Analog	GATE_POS_HYS	AD conversions are enabled when the external analog trigger input is more negative than LowThreshold. AD conversions are disabled when the external analog trigger input is more positive than HighThreshold. Hysterisis is the level between LowThreshold and HighThreshold.
Analog	GATE_ABOVE	AD conversions are enabled as long as the external analog trigger input is more positive than HighThreshold.
Analog	GATE_BELOW	AD conversions are enabled as long as the external analog trigger input is more negative than LowThreshold.
Analog	TRIG_ABOVE	AD conversions are enabled when the external analog trigger makes a transition from below HighThreshold to above. Once conversions are enabled, the external trigger is ignored.
Analog	TRIG_BELOW	AD conversions are enabled when the external analog trigger input goes from above LowThreshold to below. Once conversions are enabled, the external trigger is ignored.

Analog	GATE_IN_WINDOW	AD conversions are enabled if external analog trigger is inside the region defined by LowThreshold and HighThreshold.
Analog	GATE_OUT_WINDOW	AD conversions are enabled if external analog trigger is outside the region defined by LowThreshold and HighThreshold.
Digital	GATE_HIGH	AD conversions are enabled as long as the external digital trigger input is 5V (logic HIGH or 1).
Digital	GATE_LOW	AD conversions are enabled as long as the external digital trigger input is 0V (logic LOW or 0).
Digital	TRIG_HIGH	AD conversions are enabled when the external digital trigger is 5V (logic HIGH or ‘1’). Once conversions are enabled, the external trigger is ignored.
Digital	TRIG_LOW	AD conversions are enabled when the external digital trigger is 0V (logic LOW or ‘0’). Once conversions are enabled, the external trigger is ignored.
Digital	TRIG_POS_EDGE	AD conversions are enabled when the external digital trigger goes from 0V to 5V 9 (logic LOW to HIGH). Once conversions are enabled, the external trigger is ignored.
Digital	TRIG_NEG_EDGE	AD conversions are enabled when the external digital trigger goes from 5V to 0V (logic HIGH to LOW). Once conversions are enabled, the external trigger is ignored.

UNSIGNED *LowThreshold*

Selects the low threshold used when the trigger input is analog. Must be 0-4095 for 12-bit boards and 0-65535 for 16-bit boards. See Note.

This parameter is ignored when the trigger input is digital.

UNSIGNED *HighThreshold*

Selects the high threshold used when the trigger input is analog. Must be 0-4095 for 12-bit boards and 0-65535 for 16-bit boards. See Note.

Returns: **int** Error Code. Zero if the function is successful. Non-zero if the function fails.

Note: The value of the threshold must be within the range of the analog trigger circuit associated with the board. Please refer to board-specific information. For example, on the PCI-DAS1602/16 the analog trigger circuit handles +/-10V and therefore, a value of 0 would correspond to minus 10V whereas a value of 65535 would correspond to plus 10V.

cbStopBackground()

Description: Stops any background operation that is in progress for the specified board. This function can be used to stop any function that is running in the background. This includes any function that was started with the BACKGROUND option as well as cbCStoreOnInt() which always runs in the background.

cbStopBackground() should be also executed after normal termination of all background functions in order to clear variables and flags.

Summary: int cbStopBackground (int BoardNum)

Arguments: BoardNum - May be 0 to 99 (0 to 9 for 16 bit version of Universal Library).
Two arguments are added to the panel in HP VEE, they are:
Count - The remaining count to completion of the background scan
Data Array - Where the data from the background task is returned.

Returns: Error code or 0 if no errors

HP VEE

The cbStopBackground() function returns data in HP VEE. This is done to allow you to run a background scan process and retrieve the data after it has stopped.

VEE Programs Stopping Background Tasks Early

The red STOP button on the cbGetStatus panel must be used when stopping background processes before the scheduled completion. If instead you use the stop button on the VEE icon bar, the background process will continue to run in the background. The result of doing so and exiting VEE is undefined. Please do not fail to use the cbGetStatus() STOP button. See example programs ULAI03.VEE through ULAI06.VEE.

cbTIn()   Changed R3.3 ID

Description: Reads an analog input channel, linearizes it according to selected temperature sensor type, and returns the temperature in degrees. The CJC channel, the gain, and sensor type, are read from the InstaCal configuration file. They should be set by running the InstaCal™ configuration program.

Summary: int cbTIn (int BoardNum, int Chan, int Scale, float *TempVal, int Options)

Arguments: BoardNum - May be 0 to 99 (0 to 9 for 16 bit version of Universal Library).
Chan - Channel to read
Scale - The temperature scale for which to calculate the temperature in degrees.
TempVal - Temperature returned here
Options - Bit fields that control various options

EXPLANATION OF THE ARGUMENTS:

Chan - Input channel to read.
For EXP boards the channel number is calculated using the following formula:
A/DChan = A/D channel that mux is connected to
MuxChan = Mux board input channel number
Chan = (ADChan+1) * 16 + MuxChan (where MuxChan ranges from 0 to 15, indicating which channel on a particular bank or board)

For example, if you had an EXP16 connected to a CIO-DAS08 via the CIO-DAS08 channel 0 (remember, DAS08 channels are numbered 0,1,2,3,4,5,6 & 7), AND if you had a thermocouple connected to channel 5 of the EXP16, the value for Chan would be (0+1)*16 + 5 = 21.

Scale - Specifies the temperature scale that the input will be converted to. Choices are CELSIUS, FAHRENHEIT and KELVIN.

TempVal - The temperature in degrees is returned here. Thermocouple resolution is approximately 0.25 degrees C, depending on scale, range and thermocouple type. RTD resolution is 0.1 degrees C. If an OPENCONNECTION error occurs (for example, if a thermocouple breaks) the value returned will be -9999.0.

Options:
FILTER - The TIn applies a smoothing function to temperature readings very much like the electrical smoothing inherent in all hand held temperature sensor instruments. This is the default. When selected, 10 samples are read from the specified channel and averaged. The average is the reading returned. Averaging removes normally distributed signal line noise.

NOFILTER - If you use the NOFILTER option then the readings will not be smoothed and you will see a scattering of readings around a mean.

Returns: Error code or 0 if no errors
*TempVal - Temperature returned here

Note on CJC Channel: The CJC channel is set in the InstaCal install program. If you have multiple EXP boards, Universal Library will apply the CJC reading to the linearization formula in the following manner. First, if you have chosen a CJC channel for the EXP board that the channel you are reading is on, it will use the CJC temp reading from that channel. Second, if you have left the CJC channel for the EXP board that the channel you are reading is on to NOT SET, the library will use the CJC reading from the next lower EXP board with a CJC channel selected.

For example: If you have 4 CIO-EXP16 boards connected to a CIO-DAS08 on channel 0,1,2 and 3, and you have chosen CIO-EXP16 #1 (connected to CIO-DAS08 channel 0) to have its CJC read on CIO-DAS08 channel 7 , AND, you have left CIO-EXP16s 2, 3 and 4 CJC channels to NOT SET, then those CIO-EXP boards will all use the CJC reading from CIO-EXP16 #1, connected to channel 7 for linearization. As you can see, it is important to keep the CIO-EXP boards in the same case and out of any breezes to ensure a clean CJC reading.

A/D Range - IMPORTANT - If the EXP board is connected to an A/D that does not have programmable gain (DAS08, DAS16, DAS16F) then the A/D board range is read from the configuration file (cb.cfg). In most cases, set hardware-selectable ranges to +/-5V for thermocouples and 0-10V for RTDs. See board-specific information for your board. If the board does have programmable gains, the cbTIn() function will set the appropriate A/D range.

cbTInScan()   Changed R3.3 ID

Description: Reads a range of channels from an analog input board, linearizes them according to temperature sensor type, and returns the temperatures to an array in degrees. The CJC channel, the gain, and temperature sensor type are read from the configuration file. Use the InstaCal configuration program to change any of these options.

Summary: int cbTInScan (int BoardNum, int LowChan, int HighChan, int Scale, float DataBuffer[], int Options)

Arguments: BoardNum - May be 0 to 99 (0 to 9 for 16 bit version of Universal Library).
LowChan - Low mux channel of scan
HighChan - High mux channel of scan
Scale - CELSIUS, FAHRENHEIT or KELVIN
DataBuffer - Temperature returned here
Options - Bit fields that control various options

EXPLANATION OF THE ARGUMENTS:
LowChan/HighChan - Specify the range of channels that will be scanned.
For EXP boards these channel numbers are calculated using the following formula:
A/DChan = A/D channel that mux is connected to
MuxChan = Mux board input channel number
Chan = (ADChan+1) * 16 + MuxChan (where MuxChan ranges from 0 to 15, indicating which channel on a particular bank or board)

For example, if you had an EXP16 connected to a CIO-DAS08 via the CIO-DAS08 channel 0 (remember, DAS08 channels are numbered 0,1,2,3,4,5,6 & 7), AND if you had thermocouples connected to channels 5, 6, and 7 of the EXP16, the value for LowChan would be (0+1)*16 + 5 = 21 and the value for HighChan would be (0+1) * 16 + 7 = 23.

Scale - Specifies the temperature scale that the input will be converted to. Choices are CELSIUS, FAHRENHEIT and KELVIN.

DataBuffer[] - The temperature is returned in degrees. Each element in the array corresponds to a channel in the scan. DataBuffer must be at least large enough to hold HighChan - LowChan + 1 temperature values. Thermocouple resolution is approximately 0.25 degrees C, depending on scale, range and thermocouple type. RTD resolution is 0.1 degrees C. For most boards, if an OPENCONNECTION error occurs (for example, if a thermocouple breaks) the value returned will be -9999.0. Some boards (the CIO-DAS-TC for example - see board specific information) cannot differentiate between channels when using cbTInScan() so an OPENCONNECTION on one channel causes the entire block of data to be returned as -9999.0. If this is undesirable in your application, use the cbTIn() function instead.

Options:
FILTER - The TIn applies a smoothing function to temperature readings very much like the electrical smoothing inherent in all hand held temperature instruments. This is the default. When selected, 10 samples are read and averaged on each channel. The average is the reading returned. Averaging removes normally distributed signal line noise.

NOFILTER - If you use the NOFILTER option then the readings will not be smoothed and you will see a scattering of readings around an mean.

Range - IMPORTANT - If the EXP board is connected to an A/D that does not have programmable gain (DAS08, DAS16, DAS16F) then the A/D board range is read from the configuration file (CB.CFG). In most cases, set hardware- selectable ranges to +/-5V for thermocouples and 0-10v for RTDs. If the board has programmable gain, the cbTInScan() function will select the appropriate A/D range. See board-specific information for your board.

Returns:	Error code or 0 if no errors
	DataBuffer[] - Temperature values in degrees are returned here for each channel in scan

Note on CJC Channel: The CJC channel is set in the InstaCal install program. If you have multiple EXP boards, Universal Library will apply the CJC reading to the linearization formula in the following manner. First, if you have chosen a CJC channel for the EXP board that the channel you are reading is on, it will use the CJC temp reading from that channel. Second, if you have left the CJC channel for the EXP board that the channel you are reading is on to NOT SET, the library will use the CJC reading from the next lower EXP board with a CJC channel selected.

For example: If you have 4 CIO-EXP16 boards connected to a CIO-DAS08 on channel 0,1,2 and 3, and you have chosen CIO-EXP16 #1 (connected to CIO-DAS08 channel 0) to have its CJC read on CIO-DAS08 channel 7 , AND, you have left CIO-EXP16s 2, 3 and 4 CJC channels to NOT SET, then those CIO-EXP boards will all use the CJC reading from CIO-EXP16 #1, connected to channel 7 for linearization. As you can see, it is important to keep the CIO-EXP boards in the same case and out of any breezes to ensure a clean CJC reading.

VERY IMPORTANT NOTE

In order to understand the functions, you must read the Board-Specific Information section found in the Universal Library user's guide. The example programs should be examined and run prior to attempting any programming of your own. Following this advice will save you hours of frustration, and possibly time wasted holding for technical support.

This note, which appears elsewhere, is especially applicable to this function. Now is the time to read the board-specific information for your board. We suggest that you make a copy of that page to refer to as you read this manual and examine the example programs.

cbWinBufAlloc()

Description: Allocates a Windows global memory buffer which can be used with the scan functions() and returns a memory handle for it.

Summary: int cbWinBufAlloc (long NumPoints)

Arguments: NumPoints - Size of buffer to allocate.

EXPLANATION OF THE ARGUMENTS:

NumPoints - specifies how many data points (integers NOT bytes) can be stored in the buffer.

Returns - 0 if buffer could not be allocated or a non-zero integer handle to the buffer.

Notes: Unlike most other functions in the library, this does not return an error code. It returns a Windows global memory handle which can then be passed to the scan functions in the library. If an error occurs, the handle will come back as 0 to indicate the error.

cbWinBufFree()

Description: Frees a Windows global memory buffer which was previously allocated with the cbWinBufAlloc() function.

Summary: int cbWinBufFree (int MemHandle)

Arguments: MemHandle - A Windows memory handle

EXPLANATION OF THE ARGUMENTS:

MemHandle - This must be a memory handle that was returned by cbWinBufAlloc() when the buffer was allocated.

Returns - Error code or zero if no errors.

cbWinArrayToBuf()

Description: Copies data from an array into a Windows memory buffer.

Summary: int cbWinArrayToBuf (unsigned *DataArray, int MemHandle, long FirstPoint, long Count)

Arguments: DataArray - Array containing data to be copied
MemHandle - Handle of buffer to copy into
FirstPoint - Index to first point in buffer
Count - Number of points to copy

EXPLANATION OF THE ARGUMENTS:

DataArray - The array containing the data to be copied

MemHandle - This must be a memory handle that was returned by cbWinBufAlloc() when the buffer was allocated. The data will be copied into this buffer

FirstPoint - Index of first point in memory buffer where data will be copied to.

Count - Number of data points to copy.

Returns - Error code or zero if no errors.

Notes: This function copies data from an array to a Windows global memory buffer. This would typically be used to initialize the buffer with data before doing an output scan. Using the FirstPoint and Count argument it is possible to fill a portion of the buffer. This can be useful if you want to send new data to the buffer after a BACKGROUND+CONTINUOUS scan command has sent the old data - i.e., circular buffering.

This function is available in the Windows C library but is not required since it is possible to manipulate the memory buffer directly from a C program using the Windows GlobalLock function. This method avoids having to copy the data from an array to a memory buffer - See example below.

```
long     Count;      /* Data declarations */
unsigned *DataArray;
int   i, MemHandle;
Count = 100
MemHandle = cbWinBufAlloc (Count);
DataArray = GlobalLock (MemHandle);
cbAInScan (......,MemHandle,...);
for i=0; i<Count; i++)
printf ("%d\n", DataArray[i]);
cbWinBufFree (MemHandle);
```

cbWinBufToArray()

Description:	Copies data from a Windows memory buffer into an array.
Summary:	int cbWinBufToArray (int MemHandle, unsigned *DataArray, long FirstPoint, long Count)
Arguments:	MemHandle - Handle of buffer from which to copy data DataArray - Array to which data will be copied FirstPoint - Index to first point in buffer. Count - Number of points to copy

EXPLANATION OF THE ARGUMENTS

MemHandle - This must be a memory handle that was returned by cbWinBufAlloc() when the buffer was allocated. The buffer should contain the data that you want to copy.

DataArray - The array that the data will be copied to.

FirstPoint - Index of first point in memory buffer that data will be copied from.

Count - Number of data points to copy.

Returns - Error code or zero if no errors.

Notes: This function copies data from a Windows global memory buffer to an array. This would typically be used to retrieve data from the buffer after executing an input scan function.

Using the FirstPoint and Count argument it is possible to copy only a portion of the buffer to the array. This can be useful if you want foreground code to manipulate previously collected data while a BACKGROUND scan continues to collect new data.

This function is available in the Windows C library but is not required since it is possible to manipulate the memory buffer directly from a C program using the Windows GlobalLock function. This method avoids having to copy the data from an array to a memory buffer - See example below.

```
long     Count;      /* Data declarations */
unsigned *DataArray;
int      i, MemHandle;

Count = 100 MemHandle = cbWinBufAlloc (Count);
DataArray = GlobalLock (MemHandle);
cbAInScan (......,MemHandle,...);
for i=0; i<Count; i++)
    printf ("%d\n", DataArray[i]);
cbWinBufFree (MemHandle);
```

3 HP VEE SPECIFIC FUNCTIONS

The functions that begin with the name cbv are specific to HP VEE. No similar functions are available to language programmers. The functions are required to cope with the extra level of data management provided by VEE. The functions are presented in the same format as all other Universal Library functions, and are supported by the example programs. For example, the function cbvAInGetData() is illustrated in the ULAI examples. Other cbv functions are presented in the examples augmented by those functions.

cbvAInGetData()

Description: Retrieves analog input data from background scan tasks. It is especially useful for monitoring data from continuous background tasks. See example program ULAI06.VEE for usage.

Summary: int cbvGetAInData(int BoardNum, int LowChan, int HighChan, int Count, int *DataArray)

Arguments:	BoardNum - May be 0 to 99 (0 to 9 for 16 bit version of Universal Library).
	LowChan - First A/D channel of scan
	HighChan - Last A/D channel of scan
	Count - Number of A/D samples to collect
	DataArray - Data array to store A/D values

EXPLANATION OF THE ARGUMENTS:

BoardNum - Refers to the board number associated with the board when it was installed with the InstaCal configuration program.

Low / High Channel # - The maximum allowable channels depends on which type of A/D board is being used. For boards that have both single ended and differential inputs the maximum allowable channel number also depends on how the board is configured. For example, a CIO-DAS1600 has 8 channels for differential and 16 for single ended.

Count - Specifies to total number of A/D samples that will be collected. If more than one channel is being sampled, then the number of samples collected per channel is equal to Count / (HighChan - LowChan + 1).

DataArray - The data array must be big enough to hold at least Count number of integers.

Returns: Error code or 0 if no errors.

cbvGetCtrStatus()

Description: Returns status about background counter operation started by cbCStoreOnInt currently running. It also returns counter data. See the example program ULCT03.VEE for usage.

Summary: int cbvGetCtrStatus(int BoardNum, int *Status, int *CurCount, int CtrCount, int CtrData[])

Arguments: BoardNum - May be 0 to 99 (0 to 9 for 16 bit version of Universal Library).
Status - pointer to where status will be returned.
CurCount - current interrupt count returned to this variable.
CtrCount - number of counters on board also equal to number of elements in CtrData
CtrData - array where counts will be stored.

EXPLANATION OF THE ARGUMENTS:

BoardNum - Refers to the board number associated with the board when it was installed with the InstaCal configuration program. The specified board must have a 9513 counter.

Status - Status indicates whether or not a background process is currently executing.

CurCount - Specifies how many interrupts have taken place at the time of the call to cbvGetCtrStatus().

CtrCount - Specifies the total number of counters on the board specified by the BoardNum argument.

CtrData - The array should have an element for each counter on the board. (5 elements for CTR-05 board, 10 elements for a CTR-10). Each element corresponds to a possible counter channel. Each channel that was marked as ENABLED in the CntrControl array passed to the cbCStoreOnInt() function will be read when an interrupt occurs. The count value will be stored in the CtrData element associated with that channel.

Returns: Error code or 0 if no errors.

cbvAOutSetData()

Description: Creates and sets default values for a data array to be used with the cbAOutScan() function. See example program ULAO02.VEE for usage.

Summary: int *cbvAOutSetData(int LowChan, int HighChan, int Count, int InitialValue)

Arguments:	LowChan - First D/A channel of scan.
	HighChan - Last D/A channel of scan.
	Count - Number of D/A samples to generate.
	InitialValue - Value used to initialize each member of the constructed data array.

EXPLANATION OF THE ARGUMENTS:

Low/High Chan - The maximum allowable channel depends on which type of D/A board is being used. When using the cSBX-DDA04 board you must specify the channel range with LowChan/HighChan even if the channel number is placed in the 4 MSB's of the data values in your data buffer.

Count - Specifies the total number of D/A values that will be initialized. Most D/A boards (all except for cSBX-DDA04) do not support timed outputs. For these boards count should be set to the number of channels in the scan.

InitialValue - Specifies the value to which each array element will be set.

Returns: A two dimensional integer array of total size Count with each element initialized to IntialValue.

cbvDOutSetData()

Description: Creates and sets default values for a data array to be used with the cbDOutScan() function for boards with DOutScan capability such as the CIO-PDMA16. See example program ULDO03.VEE for usage.

Summary: int *cbvDOutSetData(int Count, int InitialValue)

Arguments: Count - Number of digital output samples to generate.
InitialValue - Value used to initialize each member of the constructed data array.

EXPLANATION OF THE ARGUMENTS:

Count - Specifies the total number of digital output values that will be initialized.

InitialValue - Specifies the value to which each array element will be set.

Returns: A two dimensional integer array of total size Count with each element initialized to IntialValue.

cbvCounterSetData()

Description: Creates and sets default values for a data array to be used with the cbCStoreOnInt() function. The cbCStoreOnInt() function may only be used with boards that have 9513 counters. See example program ULCT03.VEE for usage.

Summary: int *cbvCounterSetData(int CtrCount, int CtrControl)

Arguments: CtrCount - Number of counters on board.
CtrControl - Value used to initialize each member of the constructed data array.

EXPLANATION OF THE ARGUMENTS:

CtrCount - Specifies the total number of counter data elements that will be initialized. This number must equal the number of counters on your board. For example, 5 elements for a CTR-5 and 10 elements for a CTR-10.

CtrControl - Specifies the value to which each array element will be set. This value is either DISABLED or ENABLED.

Returns: An integer array of size CtrCount with each element initialized to either DISABLED or ENABLED as specified by the CtrControl parameter.

cbvChanMux()

Description: Extracts a one dimensional array of data from a two dimensional array of multichannel data generated by an input scan function. See example programs using cbAInScan such as ULAI02.VEE through ULAI06.VEE for usage.

Summary: int *cbvChanMux(int Channel, int DataArray[][])

Arguments: Channel - The nth channel within the DataArray.
DataArray - Data generated from an input scan function.

EXPLANATION OF THE ARGUMENTS:

Channel - Specifies which channel within the two dimensional DataArray to extract. This number may not correspond to the hardware channel. For example, if hardware channels 2, 3 and 4 were collected into DataArray, a Channel argument of 0 will extract the data for hardware channel 2 since it is the 0th element in the DataArray.

DataArray - a two dimensional array with data collected from an input scan function.
Returns: A one dimensional array containing data from a single channel.

For your notes.

For your notes

ComputerBoards
16 Commerce Blvd.
Middleboro, MA 02346
Tel: (508) 946-5100
Fax: (508) 946-9500
E-mail: info@computerboards.com
www.computerboards.com